TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MARKUS KURONEN

OHJELMISTON NYKYAIKAISTAMINEN JA UUDISTAMINEN

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 06. helmikuuta 2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Kuronen, Markus: Ohjelmiston nykyaikaistaminen ja uudistaminen
Diplomityö, 48 sivua, 0 liitesivua
Joulukuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen
Avainsanat: Luottamushenkilö rekisteri, ohjelmiston uudistaminen, käytettävyys, arkkitehtuuri, SmartGWT

Ohjelmiston uudistamisella tarkoitetaan ohjelmiston toiminnallisuuden parantamista ja ohjelmiston päivittämistä vastaamaan nykyisiä ohjelmistolle esitettyjä tarpeita. Ohjelmiston uudistamisella on myös mahdollista saada vanhalle ohjelmistolle lisää elinkaarta tai se voi olla sujuva keino lopettaa vanhan ohjelmiston elinkaari, ja siirtyä käyttämään uudistettua ohjelmistoa.

Tässä diplomityössä toteutettu työ jakautuu useaan vaiheeseen. Vaiheissa käydään läpi luottamushenkilörekisteriä, luottamushenkilörekisterin asettamia vaatimuksia, käyttöliittymän suunnittelua, ohjelmiston uudistamista sekä testausta. Jokaisessa osiossa pyritään selittämään osion tarpeellisuus ja toiminnallisuus. Kuitenkin pääpaino kussakin osiossa on ohjelmiston uudistamisessa.

Työn tulokset ovat osoittaneet, että ohjelmistoa uudistettaessa on sille varattava riittävästi aikaa ja resursseja, jotta se on mahdollista toteuttaa sujuvasti ja jotta uudistamisen tuloksena saadaan toimiva, vanhan korvaava ohjelmisto. Uudistamista ei voi tehdä vain muiden toimien ohella. Edellä mainittujen lisäksi voidaan todeta myös, että hyvällä suunnittelulla ja vanhaan ohjelmistoon tutustumisella voidaan uudistusvaiheessa säästää paljon resursseissa. Uudistamisprosessissa saattaa vanha ohjelmisto, sen osat, sekä uudet toimintatavat, ohjelmointikieli ja mahdolliset sovelluskehitykset asettaa omia rajoituksia uudistamisen toteuttamiseen. Rajoitusten lisäksi ne voivat myös vastaavasti avata uusia mahdollisuuksia ja nopeuttaa uudistamisprojektia.

Työstä voidaan päätellä, että ohjelmiston uudistamisen yhteydessä on mahdollista uudistaa koko ohjelmisto ja samanaikaisesti lopettaa vanhan ohjelmiston elinkaari, sekä tuoda markkinoille uusi ohjelmisto. Uudessa ohjelmistossa on vahan ohjelmiston hyvät ominaisuudet ja uuden ohjelmiston hienous ja toiminnallisuus. Lisäksi on mahdollisuus käyttää siirtymävaiheen ajan rinnakkain vanhaa ja uutta ohjelmistoa hukkaamatta tai siirtämättä jo olemassa olevaa dataa.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Kuronen, Markus: Software modernization and renewal
Master of Science Thesis, 48 pages, 0 Appendix pages
December 2015
Major: Software Engineering
Examiner: Professor Tommi Mikkonen
Keywords: Elected official register, usability, architecture, SmartGWT

Software renewal means that we are improving software's functionality and updating it to serve today's needs. With software renewal it is possible to have much longer life cycle or it could be fluent way to end old software life cycle and switch to use new reformed software.

This thesis is divided into several stages. In the different stages we go through elected register and its demands, planning the user interface, software reforming and software testing. In each stages we explain why this stage is important and how does it works. Mainly we try to keep main focus in to the software reforming.

Study shows, that you have to take sufficiently time and resources for software reforming without forgetting good planning and knowing reforming software. Whit those elements you are able to get well-functioning software and you can save money and time. It is also good to keep in mind than in software reforming an old software, its components, program language and frameworks may cause constraints in to the software reforming. On the other hand those may also open new possibilities and speed up software reforming.

In results we can also deduce that software renewal gives change to rework software and at the same time end old software maintenance. After that you can bring "new" software in to the customers which keeps inside it old software's good things and new software innovations. You may also able to keep old and new software in business that time that the transition from one program to another takes. In addition you may have able to user old's software databases or storages so you do not have to move or copy data anywhere.

ALKUSANAT

Tämä diplomityö on tehty Triplan Oy:ssä, jossa toimin ohjelmistosuunnittelijana. Idea työhön lähti projektista, jossa oli tavoitteena uudistaa ja nykyaikaistaa vanha, niin sanottu työpöytäsovellus (client) nykypäiväiseksi, verkossa toimivaksi sovellukseksi. Työssä tutkin ohjelmiston uudistamista projektissa, jossa uudistettavaa ohjelmaa ei voi vain kääriä. Käärimisellä tarkoitetaan sitä, että rakennetaan vanhan ohjelman ympärille rajapinta siten, että vanhaa ohjelmaa käytettään ainoastaan tämän uuden rajapinnan kautta. Ohjelman uudistamisessa on otettava huomioon erilaisia asioita, kuten useat yhtäaikaiset käyttäjät, tietoturva erinäkökulmasta sekä uudet tekniset haasteet.
Haluan kiittää Triplan Oy:tä, että sallivat työn tekemisen projektista. Lisäksi haluan kiittää työn ohjauksesta Tampereen teknillisen yliopiston professori Tommi Mikkosta.

Markus Kuronen

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

CGA	CGA on lyhenne englanninkielisistä sanoista Color Graphics Adapter. Se on ensimmäinen PC:llä värejä mahdollistanut grafiikka standardi, jonka on kehittänyt IBM vuonna 1981. Sen käytetyin/tunnetuin grafiikkatila oli 300x200 pikseliä mahdollistava resoluutio ja neljä väriä mahdollistanut väritila. Väritilan väreinä olivat aina joko musta + vihreä + punainen + ruskea (/oranssi) tai musta + violetti (magenta) + harmaa + turkoosi (/syaani /cyan) [1].
CSS	CSS (Cascading Style Sheets) on WWW-dokumenteille kehitetty tyyliohjeiden laji. CCS dokumentille voidaan määritellä useita erilaisia tyyliohjeita, jotka yhdistetään säännöstöksi. Nämä säännöstöt ehdottavat kuinka dokumentti kuvataan, eivätkä ne ole ehdottomia [2].
DBPool	Java -pohjainen tietokantayhteys apuohjelma / kirjasto, mikä tukee joutoajan hyväksi käyttöä, SQL-pyyntöjen kirjoittamista välimuistiin, yhteyden validointia [3].
EGA	EGA lyhenne muodostuu englannin kielen sanoista Enhanced Graphics Adapter. EGA:n on kehittänyt IBM ja se on julkaissut näyttöstandardin vuonna 1984. EGA näyttöstandardi on CGA:n ja VGA:n väliin sijoittuva standardi, joka sisältää 16 värin väripaletin ja 640x350 pikseliä mahdollistavan resoluution [4].
GWT	GWT on Googlen kehittämä avoimeen lähdekoodiin pohjautuva kehitys työkalupakki (development toolkit) monimutkaisten selainpohjaisten sovellusten toteuttamiseen ja optimointiin [5].
HTML	HTML (Hypertext Markup Language) on hypertekstiä, toisin sanoen hyperlinkkejä sisältävää tekstiä. Se on avoimesti standardoitu ja laajasti käytetty, ja lähes kaikki internet sivustot pohjautuvat siihen [6].
IEEE 829	Dokumentti, jonka tarkoituksena on raportoida jonkin järjestelmän tai komponentin vika, joka voi aiheuttaa kom-

	ponentin tai järjestelmän toimintaan siten, että se ei pysty suorittamaan vaadittua toimintoa [7].
JavaScript	JavaScript on dynaamisesti tyypitetty, tulkattava oliopohjainen komentosarjakieli/ohjelmointikieli. Sen avulla pystytään esimerkiksi lisäämään Web-sivuille dynaamista toiminnallisuutta ja se mahdollistaa käyttäjän interaktion selaimeen tuotetun ohjelmistokoodin /ohjelmiston kanssa [2].
Java Servlet	Pienikokoisia laitteistoriippumattomia Java-ohjelmia, jotka toimivat palvelimella. Niiden avulla voidaan laajentaa palvelimen toiminnallisuutta [8].
Käyttöskenaario	Käyttöskenaarion tarkoituksena on kuvata kuinka käyttäjä pystyy suorittamaan tietyn tehtävän tietyssä ympäristössä. Sen tavoitteena on huomioida myös tehtävän suorittamisesta mahdollisesti aiheutuvat ongelma ja erikoistilanteet [9].
Käyttötapaus	Käyttötapauksen tarkoituksena on määritellä järjestelmän perustoimenpidesarjat ja vaihtoehtoiset sarjat, jotka järjestelmä voi suorittaa vuorovaikuttaen käyttäjän kanssa. Käyttötapausten on tarkoitus ohjata järjestelmän kehityksessä vuorovaikutuksen suunnittelua, arkkitehtuurin luontia ja testaamista, testitapausten ja -kierrosten määrittämistä, iteraatioiden suunnittelua ja käyttöohjeiden luontia [10].
Luottamushenkilö rekisteri	Henkilörekisteri, jossa ylläpidetään rekisteriselosteen mukaisia tietoja luottamushenkilöistä [11].
Rekisteriseloste	Laaditaan kaikista henkilörekistereistä. Siitä ilmenee, kuka on henkilötietojen käsittelystä vastaava rekisterin pitäjä, sekä mitä henkilötietoja rekisterissä on ja mihin niitä käytetään, sekä mihin niitä säännönmukaisesti luovutetaan. Lisäksi siitä ilmenee suojauksen periaatteet [12].
RIA	RIA (Rich internet application) eli suomeksi käännettynä rikas internet sovellus. Tarkoittaa Web-sovellusta, jonka ominaisuudet ja toiminnallisuus on kuin perinteisellä työpöytäsovelluksella [13].
SmartGWT	Java -pohjainen Googlen kehittämä ohjelmisto kehys, joka pohjautuu GWT:hen. Sillä pystyy kohtalaisen helposti te-

	kemään näyttäviä web-käyttöliittymiä käyttämällä Java ohjelmointikieltä [13].
Sovelluskehys	Ohjelmoinnin apuväline, joka on valmistettu nopeuttamaan ohjelmistotuotteiden kehitystä. Ohjelmisto kehys tarjoaa valmiita komponentteja, joita voidaan hyödyntää sovelluskehityksessä [14].
Swing	Swing on graafisen käyttöliittymän toteuttamiseen tarkoitettu sovelluskehys, joka on kirjoitettu kokonaisuudessa Javalla [15].
TDD	Ohjelmistokehityksen käytäntö, jossa yksikkötesti kirjoitetaan ennen yksikön toteuttavaa koodia [16].
VGA	VGA on lyhenne sanoista Video Graphics Array. Se on PC:lle toteutettu värillisen grafiikan esittämisen mahdollistava standardi. Sen on kehittänyt IBM vuonna 1987. Sen tunnetuimmat/käytetyimmät grafiikkatilat ovat 640x480 resoluution ja 16 väriä, sekä 320x20 resoluution ja 256 värin tilan muodostavat tilat. VGA:ssa esitettävässä grafiikassa värit voidaan valita 262144 värin paletista [17].
Visual Basic	Visual Basic on Microsoftin kehittämä ohjelmointikieli, joka soveltuu erikokoisten ohjelmistojen laadintaan [18].
Web-ohjelma	Web-ohjelma on ohjelma, jota suoritetaan internet-selaimessa. Sen toiminta perustuu internet selaimessa suoritettaviin tekniikoihin, kuten HTML, CSS, JavaScript yms. ja palvelimella suoritettaviin tekniikoihin, kuten Java Servlet [19].

1 JOHDANTO

Ohjelmistot vanhenevat, kun aikaa kuluu riittävästi ja ohjelman käytön tarpeet muuttuvat tai lisääntyvät. Ohjelmistojen vanheneminen johtuu erilaisista muutos-, kehitys- ja muokkaustoimista, jotka muuttavat ohjelmiston rakennetta. Rakenteen muuttuminen puolestaan saattaa vaikuttaa ohjelman monimutkaistumiseen. Toisin sanoen ohjelmallinen arkkitehtuuri rapistuu ajan ja muutostöiden myötä. Tästä johtuen ohjelman ylläpidettävyys ja kunnossapito-kustannukset nousevat niin koviksi, ettei normaali ylläpito ole enää kenenkään kannalta järkevää. Edellä mainittu ohjelmien vanheneminen noudattaa kuutta lakia, jotka Volady ja Lehman esittivät kirjassaan vuonna 1976 [20]. Alla esitettynä edellä mainitut Lehmannin ohjelmien evoluutiolain kuusi kohtaa [21]:

1. Jatkuvan muutoksen laki – Käytetty ohjelma käy läpi jatkuvia muutoksia, kunnes se todetaan päivityskelvottomaksi.

2. Lisääntyvän monimutkaisuuden laki – Ohjelma käy aikaa myöden entistä monimutkaisemmaksi, mikäli erityistä huomiota ei kiinnitetä tämän estämiseen.

3. Itseohjautuvuus – Järjestelmän kehityskulku määräytyy ”liiketoiminnan” palauteprosessin mukaisesti.

4. Organisatorinen vakaus – Ohjelman kehitysnopeus on suunnilleen vakio sen elinaikana, eikä se riipu kehitykseen varatuista resursseista

5. Tuttuuden säilymisen laki – Inkrementaalinen muutos on suunnilleen vakio järjestelmän eliniän aikana kussakin julkaisussa.

6. Jatkuva kasvu – Ohjelman toiminnallisen sisällön pitää kasvaa tai muuttua ohjelmiston eliniän ajan, jotta käyttäjätyytyväisyys säilyisi.

Myös ohjelmien elinkaareen kohdistuvien kustannusten jakautuminen näyttäisi noudattavan joiltakin osin Lehmannin lakeja. Kustannusjakaumassa suunnittelu- ja implementointivaiheessa ohjelman kustannukset nousevat korkeimmilleen ohjelman koko elin-

kaaren aikana. Tämän jälkeen kustannukset pysyvät suhteellisen vakaina ja tasaisina, kuitenkin lievästi nousujohteisena. Nousujohteisuus jatkuu suhteellisen tasaisena, kunnes kustannukset ajan saatossa lähtevät taas uuteen jyrkempään nousuun, kuten kuvasta 1 ohjelman elinkaaren kustannuskehitys on nähtävissä.

Kuva 1 Ohjelman elinkaaren kustannuskehitys [20]

Näin ollen Lehmannin lait ja ohjelmiston kustannusjakauma puoltavat sitä, että ohjelman ollessa elinkaarensa päässä, se on joko haudattava tai sen elinkaarta on pyrittävä jatkamaan jollakin tavalla. Tässä vaiheessa ohjelmiston elinkaarta ohjelmiston uudistaminen saattaa olla paras vaihtoehto, riippuen ohjelmiston käyttöasteesta ja markkinoiden tarpeista.

Tämän diplomityön tarkoituksena on selvittää mitä on ohjelmiston uudistaminen, mitä siihen sisältyy, mitä sen tekemisessä pitää ottaa huomioon ja miten sen voi tehdä. Työn tavoitteena on uudistaa ja nykyaikaistaa luottamushenkilörekisterisovellus, mikä on palvellut jo oman aikansa.

Ohjelmiston uudistamisprojektiin lähdettiin, koska vanha ohjelmisto oli saavuttanut jo oman elinkaarensa pään. Tähän tulokseen oltiin tultu siksi, että käyttäjät halusivat päästä eroon laiteriippuvuuksista ja paikkariippuvuuksista. Lisäksi toiveena oli saada ohjelmisto verkkoon, jotta päästään eroon työasemakohtaisuudesta. Ohjelmiston verkkoon siirtämisen ansiosta myös ylläpito ja huoltotoimenpiteet helpottuisivat huomattavasti, kun niitä verrataan client sovelluksen kanssa tehtäviin toimenpiteisiin. Edellä mainittujen lisäksi oli toivottu melko paljon myös uusia ominaisuuksia, joita ei enää kannattanut lähteä toteuttamaan vanhaan ohjelmistoon.

Tämän diplomityön toisessa luvussa käydään läpi luottamushenkilörekisteriä, sen rakennetta ja toimintaa. Luvun käydään läpi mitä tarkoittaa rekisteriseloste, mikä se on ja mitä se tarkoittaa. Tutustutaan henkilörekisteriin ja henkilötietolakiin, mitä ne tarkoittavat ja mitä niiden vuoksi on otettava huomioon. Lisäksi tarkastellaan mitä tarkoitetaan

henkilötietojen käsittelyä koskevilla periaatteilla ja minkä takia niihin on hyvä perehtyä. Lisäksi käydään läpi mitä tarkoitetaan luottamushenkilöllä ja mitä luottamushenkilörekisteri tarkoittaa, mihin ja miten sitä käytetään, sekä ketkä sitä käyttävät.

Luvussa kolme perehdytään käyttöliittymiin yleisellä tasolla ja käydään läpi erilaisia käyttöliittymiä, kuten komentopohjainen käyttöliittymä, tekstipohjainen käyttöliittymä ja graafinen käyttöliittymä. Lisäksi esitellään mitä ne ovat ja miten niitä on aikojen saatossa käytetty ja miten ne ovat vaikuttaneet tietotekniikan ja ohjelmistotekniikan kehittymiseen. Luvussa tarkastellaan myös käyttöliittymäsuunnittelua ja siihen liittyviä perusteita sekä esitetään millainen suunnitteluprosessi käyttöliittymäsuunnittelu on. Luvussa tarkastellaan myös käyttöliittymää käytettävyyden näkökulmasta. Käydään läpi käytettävyyden historiaa, jonka jälkeen tarkastellaan käytettävyyttä yleisellä tasolla, mitä se on ja mitä se tarkoittaa. Lopuksi tarkastellaan käytettävyyttä käyttöliittymässä, mitä se on ja miten sitä voi kehittää.

Neljännessä luvussa tarkastellaan, minkälaisia vaiheita ohjelmiston uudistamiseen yleensä liittyy. Luvussa tarkastellaan myös millaisten abstraktiotasojen kautta uudistaminen tapahtuu ja mikä taso on riittävä ohjelmiston uudistamisen kannalta. Lisäksi käydään läpi ohjelmiston uudistamisessa käytettäviä menetelmiä ja mahdollisia riskejä, joita uudistamisprosessista voi aiheutua.

Viidennessä luvussa käydään läpi uudistamisprosessia, mitä se tarkoittaa käytännössä ja millainen on uudistettava ohjelmisto sekä miten uudistamisprosessi etenee. Lopuksi tarkastellaan mitä menetelmiä ja toimintatapoja siinä käytettiin.

Kuudennessa luvussa tarkastellaan sovelluskehystä, mikä on sovelluskehys ja mikä sen tarkoitus on. Luvussa kerrotaan, mihin sovelluskehystä voidaan käyttää ja miten se toimii. Luvun lopussa tarkastellaan, mitä huonoja puolia ohjelmistokehyksen käyttämiseen voi liittyä tai mitä siitä voi aiheutua.

Seitsemännessä luvussa tarkastellaan SmartGWT sovelluskehystä. Luvussa pyritään kuvaamaan mikä on SmartGWT sekä miten sitä käytetään ja miten se toimii. Edellä mainittujen lisäksi tarkastellaan, millaisella arkkitehtuurilla sovelluskehys on tehty.

Kahdeksannessa luvussa tutustutaan uudistettuun ohjelmistoon tarkemmin. Luvussa käydään läpi muun muassa uudistamisprosessin tavoitteet ja lähtökohdat. Tutustutaan uudistamisprosessin vaatimaan määrittelyyn ja suunnitteluun, sekä prosessissa käytettyyn ohjelmistokehykseen (SmartGWT). Lisäksi käydään läpi ohjelmiston kehityksessä käytetyt ohjelmointikielet ja ohjelmistoon valittu arkkitehtuuri ja sen rakenne.

Yhdeksännessä luvussa tarkastellaan ohjelmiston testaamista perehtymällä siihen, mitä testaus on, miksi sitä pitäisi tehdä, milloin se on riittävää, mitä tasoja se kattaa ja miksi sitä tehdään. Tässä luvussa tarkastellaan mitä eri tasoja testaukseen kuuluu ja mitä nämä tasot pitävät sisällään. Lisäksi tarkastellaan milloin testaus voidaan todeta riittäväksi, jotta ohjelmiston virhetaso on halutulla tasolla. Tässä luvussa kuvataan mitä dokumentteja testauksessa voi syntyä, sekä käydään läpi erilaisia testausmenetelmiä. Lopuksi vielä luodaan katsaus siitä, miten uudistettavassa ohjelmistossa testaus suoritettiin.

Kymmenennessä ja viimeisessä luvussa on yhteenveto tehdystä työstä. Luvussa tarkastellaan mitä tämä diplomityö piti sisällään ja mitä asioita siinä käsiteltiin. Minkälaisia

johtopäätöksiä ohjelmiston uudistamisesta voidaan vetää, kun sitä tarkastellaan uudistamisprosessin jälkeen. Mihin ohjelmiston uudistamisessa kannattaa yleensä kiinnittää huomiota ja miten uudistamisprosessissa kannattaisi edetä. Lisäksi tarkastellaan mitkä olivat tämän työn tavoitteet, päästiinkö projektin aikana asetettuihin tavoitteisiin ja oliko lopputulos odotuksen mukainen, sekä onnistuttiinko uudistamisprosessissa. Luvun loppupuolella käydään vielä läpi sellaisia asioita, joita uudistamisprosessissa olisi voinut tehdä toisin, jos uudistaminen aloitettaisiin nyt uudelleen. Viimeiseksi tarkastellaan vielä jäikö jotakin tekemättä ja miten uudistettavaa ohjelmistoa voisi tulevaisuudessa parantaa. Lisäksi tarkastellaan mitä jatkosuunnitelmia ohjelmiston kehittämiseen on, mitä on tarkoitus tehdä seuraavaksi ja mitä uudistamisprosessin aikana opittiin.

2 LUOTTAMUSHENKILÖREKISTERIÄ KOSKEVA LAINSÄÄDÄNTÖ JA OHJAUS

Tässä luvussa tutustutaan henkilötietolakiin ja kerrotaan, milloin henkilötietolakia tulee noudattaa ja käyttää, sekä mitä tarkoitetaan henkilötietojen käyttöä koskevilla periaatteilla. Tutustutaan henkilörekisteriin ja pyritään kartoittamaan mitä tarkoitetaan henkilörekisterillä. Käydään läpi mitä tarkoitettaan rekisteriselosteella, milloin sellainen pitää olla ja mitä sillä tehdään. Lisäksi kerrotaan mitä ovat luottamushenkilö, luottamushenkilörekisteri sekä mihin luottamushenkilörekisteriä käytetään.
Tämän luvun tekstit on osin otettu melko suoraan lähteistään, koska teksti on suurilta osin lakiin pohjautuvaa tekstiä ja sitä on hankala kertoa omin sanoin muuttamatta sen alkuperäistä tarkoitusta. Lähteinä on käytetty (löytyvät myös lähdeluettelosta): [12], [22], [23], [24], [25], [11].

2.1 Henkilötietolaki

Henkilötietolain (1999/523) tarkoituksena on toteuttaa yksityiselämän suojaa ja muita yksityisyyden suojaa turvaavia perusoikeuksia henkilötietoja käsiteltäessä. Lisäksi sen on tarkoitus edistää hyvän tietojenkäsittelytavan kehittämistä ja noudattamista [22]. Henkilötietolakia tulee soveltaa sellaisten henkilötietojen käsittelyyn, jossa rekisterin pitäjän toimipaikka on Suomen alueella tai Suomen oikeuden ulottuvuuden piirissä. Lakia sovelletaan myös silloin, kun rekisterinpitäjällä ei ole toimipaikkaa Euroopan unionin jäsenvaltioiden alueella, mutta rekisterin pitäjä käyttää henkilötietojen käsittelyyn Suomessa sijaitsevia laitteita muuhun tarkoitukseen, kuin henkilötietojen siirtoon suomen alueen kautta [22].
Henkilötietolakia tulee noudattaa aina, kun käsitellään jonkun henkilön henkilötietoja. Henkilötietolakia sovelletaan henkilötietojen automaattisen käsittelyn yhteydessä. Tämän lisäksi henkilötietolakia sovelletaan aina, kun henkilötiedot muodostavat tai niiden on tarkoitus muodostaa jonkin asteinen henkilörekisteri tai sen osa. Henkilötietolaki ei kuitenkaan kosketa henkilötietojen käsittelyä, jota henkilö suorittaa henkilökohtaisiin tai niihin verrattaviin yksityisiin tarkoituksiin [22].
Henkilötietolaissa henkilötiedolla tarkoitetaan kaikkia henkilöä, henkilön omaisuutta tai elinolosuhteita kuvaavia merkintöjä, jotka voi tunnistaa henkilön tai hänen perhettä tai hänen kanssa samassa taloudessa eläviä koskevaksi. Henkilötietojen käsittelyllä taas

tarkoitetaan henkilötietojen keräämistä, tallettamista, järjestämistä, käyttämistä, siirtämistä, luovuttamista, säilyttämistä, muuttamista, yhdistämistä, suojaamista, poistamista, tuhoamista sekä muita vastaavia henkilötietoihin kohdistuvia toimenpiteitä. Henkilörekisterillä tarkoitetaan henkilötietojoukkoa, jota käsitellään automaattisen tietojenkäsittelyn avulla tai, mikä on järjestetty kortistoksi, luetteloksi tai näihin verrattaviin siten, että henkilö koskevat tiedot voidaan löytää helposti ja vähäisillä kustannuksilla. Rekisterin ylläpitäjällä tarkoitetaan yhtä tai useampaa henkilöä, yhteisöä, laitosta tai säätiötä, jonka käyttöön rekisteri perustetaan ja jolla on oikeus määrätä henkilörekisterin käytöstä tai jonka tehtäväksi rekisterin ylläpito on laissa määrätty. Rekisteröity henkilö on henkilö, jota henkilörekisterissä oleva tieto koskettaan. Sivullisella taas tarkoitetaan jotakin muuta henkilö, yhteisöä, laitosta tai säätiötä kuin rekisteröityä, rekisterinpitäjää henkilötietojen käsittelijää [22].

2.2 Henkilötietojen käsittelyä koskevat periaatteet

Henkilötietojen käsittelyä koskevat periaatteet on määritelty henkilölaissa. Näitä periaatteita on huolellisuusvelvoite, henkilötietojen käsittelyn suunnittelu, käyttötarkoitussidonnaisuus, käsittelyn yleiset edellytykset, tietojen laatua koskevat periaatteet sekä rekisteriseloste.

Huolellisuusvelvoitteella tarkoitetaan sitä, että rekisterin pitäjän täytyy käsitellä henkilötietoja laillisesti, noudattaa huolellisuutta ja hyvää tietojenkäsittelytapaa sekä muuta sellaista toimintaa, mikä ei vaaranna rekisteröidyn yksityiselämän suojaa tai muta yksityisyyden suojaa turvaavia perusoikeuksia rajoiteta ilman laissa määrättyä perustetta. Tämä sama velvollisuus koskee myös sellaista henkilöä, joka itsenäisenä elinkeinon- tai toiminnan harjoittajana toimii rekisterin pitäjän alaisena. Henkilötietojen käsittelyn suunnittelulla tarkoitetaan sitä, että henkilötietojen käsittelyn tulee olla asiallisesti perusteltua rekisterin pitäjän toiminnan kannalta. Henkilötietojen käsittelyn tarkoitukset sekä henkilötietojen säännön mukainen hankita ja luovutus on määriteltävä ennen kuin henkilötietojen kerääminen aloitetaan tai muodostetaan henkilörekisteri. Henkilötietojen käsittelyn tarkoitus pitäisi määritellä siten, että siitä ilmenee millaisen rekisterinpitäjän tehtävien hoitamiseen henkilötietoja on tarkoitus käsitellä. Käyttötarkoitussidonnaisuudella tarkoitetaan sitä, että henkilötietoja saa käyttää vai sellaisella tavalla mikä ei ole henkilötietojen käsittelyn suunnittelu pykälässä (6 §) mainittujen seikkojen kanssa yhteensopimatonta. Tietojen laatua koskevilla periaatteilla tarkoitetaan sitä, että käsiteltävien henkilötietojen pitää olla tarkoituksen kannalta tarpeellisia. Lisäksi rekisterin pitäjän pitää huolehtia siitä, ettei virheellisiä, epätäydellisiä tai vanhentuneita henkilötietoja käsitellä rekisterissä tätä kutsutaan myös virheettömyysvaatimuksena. Rekisteri seloste tarkoittaa sitä, että rekisterin pitäjän on tehtävä henkilörekisteristä rekisteri seloste mistä käy ilmi rekisterin pitäjän ja tarvittaessa edustajan nimi ja yhteystiedot, henkilötietojen

käsittelyn tarkoitus, kuvaus rekisteröityjen ryhmästä tai ryhmistä, tieto mihin tietoja säännön mukaisesti luovutetaan ja siirretäänkö niitä Euroopan unionin tai Euroopan talousalueen ulkopuolelle sekä kuvaus rekisterin suojaus periaatteista. Lisäksi rekisterin pitäjä on velvollinen pitämään rekisteri selostetta kaikkien saatavilla. Tästä velvoitteesta voidaan kuitenkin poiketa, jos se on välttämätöntä valtion turvallisuuden, puolustuksen tai yleisen järjestyksen ja talouteen liittyvän valvontatehtävän vuoksi [22].

2.3 Henkilörekisteri

Henkilörekisterillä tarkoitetaan sellaista ylläpidettävää henkilöitä koskettavaa rekisteriä, mikä käyttötarkoituksensa vuoksi muodostaa henkilötietoja sisältävän tietojoukon, jota käsitellään joltakin osin tai kokonaan automaattisen tietojenkäsittelyn avulla. Mikäli sen sisältö on järjestetty kortistoksi, luetteloksi tai muuksi näihin verrattaviksi rakenteiksi siten, että henkilöä koskevat tiedot on helposti löydettävissä järjestelmästä ilman kohtuuttomia kustannuksia [22]. Näin ollen voidaankin sanoa, että henkilörekisterin muodostavat sellaiset henkilötietoja sisältävät tietojoukot, joita käsitellään tietotekniikan avulla. Jos kyseessä on kortisto, luettelo tai jokin muu vastaavan rakenteen omaava tietojoukko, josta henkilötiedot voidaan löytää helposti ja halvalla.

2.4 Rekisteriseloste

Rekisteriseloste on dokumentti, joka tulee laatia kaikista henkilörekistereistä. Siitä pitää ilmetä tiedot siitä, kuka on henkilötietojen käsittelystä vastaava rekisterin pitäjä, mitä henkilötietoja rekisterissä pidetään, mihin ja miten niitä käytetään, sekä mihin tietoja säännönmukaisesti mahdollisesti luovutetaan. Lisäksi siitä pitää ilmetä suojauksen periaatteet [12].
Rekisteriseloste tulee säilyttää sellaisessa rekisterin pitäjän toimipaikassa, minne jokaisella henkilöllä on oikeus päästä tarkastelemaan rekisteriselostetta. Mikäli rekisterin pitäjällä on useita toimipaikkoja samalla tai eri paikkakunnilla, on rekisteriseloste oltava saatavilla jokaisesta rekisterin ylläpitäjän toimipaikasta. Mikäli rekisterin pitäjä käyttää palvelua tietoverkossa, tulee rekisteriseloste liittää saataville verkkopalvelun yhteyteen. Halutessaan jokainen henkilö voi pyytää rekisterinpitäjältä rekisteriselosteet tarkasteltavakseen. Jos rekisterinpitäjä tai rekisterinpitäjän edustaja kieltäytyy näyttämästä rekisteriselostetta, voi henkilö olla yhteydessä asiasta tietosuojavaltuutettuun [12].

2.5 Luottamushenkilö

Luottamushenkilö on henkilö, joka on valittu yleensä vaaleilla. Kyseinen henkilö edustaa välillisesti demokratian periaatteiden mukaisesti toisia henkilöitä. Luottamushenkilöillä on keskeinen asema edustukselliseen demokratiaan pohjautuvassa kunnallisessa päätöksen teossa. Luottamushenkilön kriteerit määritellään kuntalaissa, jonka mukaan luottamushenkilöitä ovat henkilöt, jotka ovat varavaltuutettuja tai valtuutettuja, kunnan toimielimiin valittu jäsen tai varajäsen, kunnan kuntayhtymän toimielimiin valitsemat jäsenet ja varajäsenet sekä kaikki muut henkilöt, jotka toimivat kunnan luottamustoimissa. Luottamushenkilöiksi ei kuitenkaan luokitella tilintarkistajia, osakeyhtiön tai muun yksityisoikeudellisen yhteisön hallituksen tai muun toimielimen jäsentä. Luottamushenkilöksi valittavan henkilön on täytettävä kuntalaissa säädetyn yleisen vaalikelpoisuuden edellytykset. Luottamushenkilölle ominaisia tunnusmerkkejä on muun muassa pakollisuus, erottamattomuus, määräaikaisuus, vaalikelpoisuus, virkavastuu ja poliittinen vastuu. Lisäksi luottamushenkilöllä on eettisiä vastuita, joista on myös säädetty kuntalaissa. Esimerkiksi luottamushenkilön on käyttäydyttävä siten, että hän hoitaa toimensa moraalisia ja eettisiä arvoja noudattaen [24].

Kunnan luottamushenkilöt voivat toimia myös valtion toimielimissä. Yleensä kunta valitsee luottamushenkilöitä valtion toimielimiin erilaisiin virkoihin. Tällaisia virkoja on muun muassa käräjäoikeuksien lautamiehet ja kiinteistötoimitusten uskotut miehet. Vaikka kyseiset luottamusmiehet toimivat valtion toimielimissä sovelletaan heihin kuitenkin kunnan luottamushenkilöitä koskevia säädöksiä, ellei erityislaissa muuta säädetä [24].

2.6 Luottamushenkilörekisteri ja sen käyttö

Luottamushenkilörekisteriksi kutsutaan henkilörekisteriä, josta nähdään luottamushenkilönä toimivan henkilön perustiedot. Perustietoihin luokitellaan usein Tietojen salaisuus ja julkisuus, luottamushenkilönä toimivan henkilön arvo/ammatti, henkilötunnus, etunimet, sukunimi, työpaikka, lähiosoite, postitoimipaikka, postinumero, puhelin, työpuhelin, oma sähköpostiosoite, kotisivujen osoite, toimielin, puolue, pankin nimi (määritellään aina salaiseksi), tilinumero (määritellään aina salaiseksi), pankkitunniste (määritellään salaiseksi), sekä päiväys, jolloin tiedot on kirjattu järjestelmään [11].

Luottamushenkilörekisteriä ylläpitää yleensä rekisterin ylläpitäjä, ja luottamushenkilön on itse huolehdittava siitä, että tiedot pysyvät ajan tasalla. Esimerkiksi osoitteen muutokset, tilitietojen muutokset ja muut vastaavat tiedot tulee luottamushenkilön ilmoittaa kirjallisesti luottamushenkilön perustietolomakkeella luottamushenkilörekisterin ylläpitäjälle [25]. Luottamushenkilörekisteriä käytetään hallinnoimaan tietoja toimineista toimielimistä toimikausista ja niissä toimineista luottamushenkilöistä. Luottamushenkilö-

rekisterin avulla tiedot on saatavilla helposti ja keskitetysti yhdestä paikasta ja näin ollen tietoihin ja henkilöihin pystytään ottamaan yhteyttä suhteellisen helposti.

2.7 Käyttötarkoitus ja käyttäjäryhmät

Luottamushenkilörekisterin käyttötarkoituksena on yleensä pitää luetteloa tai listaa henkilöistä ja henkilötiedoista, jotka toimivat luottamustehtävissä jossakin organisaatiossa, kaupungissa, kunnassa tai toimielimessä. Luottamushenkilörekisteriin kirjataan yleensä luottamushenkilön tiedot, kuten sukunimi, etunimi, kotikunta, ammattinimike, sukupuoli, syntymäaika, sähköposti, lähiosoite, postitoimipaikka ynnä muut luottamushenkilöä koskettavat tiedot. Rekisteristä on nähtävissä myös luottamushenkilön puoluehistoria ja aikaisemmat luottamustoimet, jotka ovat järjestelmään syötetty, kuitenkin pois lukien arkaluontoiset tiedot, joita on muun muassa rotu tai etninen alkuperä, henkilön yhteiskunnallinen, poliittinen, uskonnollinen vakaumus, rikokset tai muu vastaava, henkilön terveydentilaa koskevat määritteet, seksuaalinen suuntautuneisuus ynnä muut vastaavat asiat. Edellä mainitut asioita saa kuitenkin käyttää henkilörekisterissä, jos niiden käyttöön on saatu asianomaisen tietoinen suostumus tai rekisteröidyn henkilön elintärkeän edun suojaamiseksi tai jos tietojen käsittelystä säädetään laissa ja käsittely johtuu välittömästi laissa säädetyistä tehtävistä.

Tässä diplomityössä uudistettavana ja kehitettävänä oleva luottamushenkilörekisteri ohjelmisto on tarkoitettu käytettäväksi kaikkiin sellaisiin toimielimiin, joissa on tarvetta pitää kirjaa luottamushenkilöiden tai yleensä henkilöiden tiedoista, toimikausista, toimihistoriasta ja puoluehistoriasta. Sillä ohjelmaan pystytään kirjaamaan luottamushenkilön tiedot, puoluehistoria, sekä toimihistoria. Kyseisessä ohjelmistossa pidetään kirjaa myös niistä toimielimistä ja toimikausista, joissa kyseinen luottamushenkilö on toiminut aiemmin toimiessaan luottamustehtävissä. Lisäksi ohjelmiston on myöhemmässä kehitysvaiheessa tarkoitus mahdollistaa siihen liitettävien rajapintojen kautta luottamushenkilöiden tietojen tarjoamisen myös muihin palveluihin ja ohjelmistoihin. Ohjelmistossa on myös mahdollisuus rajata tarjottavien tietojen saatavuus, jotta kaikkia tietoja ei anneta kolmansien osapuolien nähtäväksi.

Työssä uudistettavana olevan luottamushenkilörekisterin ensisijaisia käyttäjä kohteita ovat kunnat, kaupungin, seurakuntayhtymät, seurakunnat ja kaikki sellaiset yritykset ja organisaatiot, joissa toimii luottamushenkilöitä. Ohjelmiston käyttö on myös mahdollista sellaisille organisaatioille ja yrityksille, joilla on tarkoitus pitää rekisteriä ja yhteystietoja jäsenistään, työntekijöitä tai muista haluamistaan henkilöistä.

3 KÄYTTÖLIITTYMÄ

Käyttöliittymä (user interface) on osa tietotekniikkaa, tietokoneohjelmistoja ja niiden kehitystä. Ne ovat komponentteja, joiden kanssa käyttäjät ovat eniten tekemisissä. Atk-sanakirjassa käyttöliittymää pyritään määrittelemään siten, että ne ovat sellaisia välineitä ja toimintoja, joiden avulla käyttäjän on mahdollista olla vuorovaikutuksessa ohjelmiston tai laitteiston kanssa interaktiivisesti [26]. Toisin sanoen käyttöliittymällä tarkoitetaan käyttöjärjestelmän, ohjelman tai laitteen osaa, jonka avulla käyttäjän on mahdollista olla interaktiivisessa vuorovaikutuksessa näiden kanssa. Interaktiivisella vuorovaikutuksella tarkoitetaan tässä tapauksessa, että käyttäjä voi syöttää tai vastaanottaa syötteitä ohjelmalta tai sen osalta. Tämän määritelmän mukaan käyttöliittymäksi voidaankin luokitella kaikki sellaiset asiat, joilla voidaan olla jonkinasteisessa vuorovaikutussuhteessa ohjelmiston tai sen laitteiston kanssa. Sellaisia asioita ovat esimerkiksi hiiri, näppäimistö, peliohjain, puhelin, Microsoft Word tekstinkäsittelyohjelman näkyvä osa, Microsoft Windowsin työpöytä, internetselain, konsoli- tai tietokonepeli, televisio tai jokin muu vastaava laite tai ohjelmiston osa, joka sopii edellä mainittuun määritelmään.
Ohjelmistojen käyttöliittymät voidaan karkeasti jakaa kolmeen erilliseen pääryhmään, näitä ovat komentopohjaiset, tekstipohjaiset ja graafiset käyttöliittymät. Nykyisin käytettävistä käyttöliittymistä suurin osa on graafisia käyttöliittymiä. Varsinkin silloin, jos ne ovat suoranaisesti osoitettu kuluttajille. Selkeästi vähemmistönä ovat ”vanhanaikaiset” komentopohjaiset ja tekstipohjaiset käyttöliittymät, tällaisia käytetään nykyisin pääosin vain pieninä apuohjelmina tai tekemään joitakin yksinkertaisia operaatioita. Ohjelmiston käyttöliittymästä puhuttaessa pitää muistaa huomioida myös sellainen seikka, ettei sekoiteta keskenään käyttöliittymää ja käyttöjärjestelmää. Esimerkiksi Microsoft Windows tai Linux ovat tietokoneen käyttöjärjestelmiä, jotka sisältävät käyttöliittymän osia tai kokonaisia käyttöliittymiä. Usein käyttöjärjestelmän ja käyttöliittymän pystyy erottamaan toisistaan siitä, että käyttöjärjestelmä ohjaa koko tietokoneen laitteistoa ja ohjelmistoja, kun taas käyttöliittymän tarkoituksena on tarjota käyttäjälle yhteys tietokoneen resursseihin, tietoihin ja ohjelmiin tai vastaavasti ohjelmiston tietoihin, käyttäen hyväksi käyttöjärjestelmän tai ohjelmiston tarjoamaa alustaa [27].

3.1 Käyttöliittymätyypit

Tässä kohdassa käsitellään käyttöliittymätyyppejä. Tarkastelun kohteena ovat komentopohjainen, tekstipohjainen ja graafinen käyttöliittymä. Käyttöliittymistä pyritään kerto-

maan mitä ne tarkoittavat, missä niitä on käytetty ja mitä ne pitävät sisällään. Samalla tarkastellaan, mitkä vaiheet ovat johtaneet niiden kehitykseen ja käytetäänkö niitä vielä nykyään.

3.1.1 Komentopohjainen käyttöliittymä

Komentokäyttöliittymä (command line interface) periytyy reikäkorttiajoilta, jolloin tietokoneen käyttö oli hyvin suoraviivaista. Tuolloin erilaisten operaatioiden suorittaminen oli toistuvaa ja vuorottelevaa toimintaa. Komentokäyttöliittymä oli tietokoneessa se komponentti, mikä mahdollisti tietokoneen käyttämisen ilman reikäkortteja. Samalla tämä kyseinen tapa nopeutti käyttäjän ja tietokoneen vuorovaikusta huomattavasti, koska komennot ja tulosteet voitiin suorittaa samalla välineellä lähes reaaliajassa. Tyypillisin ja ehkä myös tunnetuin komentokäyttöliittymällä ajettava ohjelma on komentotulkki. Nykyisin komentokäyttöliittymästä käytetään yleisesti nimityksiä konsoli, terminaali tai pääte, esimerkiksi Microsoft windowsin käyttäjät tuntevat sen lyhenteestä cmd eli Command prompt. Nykyisin komentokäyttöliittymää käytetään vain erikoistapauksissa sen tuoman selkeyden ja pienten resurssivaatimusten takia [28]. Lisäksi käyttötarve voi olla pienten sovellusten ajamisessa, joille ei ole tehtynä erillistä käyttöliittymää.

3.1.2 Tekstipohjainen käyttöliittymä

Tekstipohjainen käyttöliittymä (Text User Interface) tai puoligraafinen käyttöliittymä on sellainen käyttöliittymä, jonka toteuttamiseen tai näyttämiseen ei käytetä näytönohjaimen graafisia ominaisuuksia hyväksi, vaan kaikki käyttöliittymässä näkyvä on tekstiä. Tekstipohjainen käyttöliittymä on niin sanottu välivaiheen käyttöliittymä, siirryttäessä täysin komentopohjaisesta käyttöliittymästä täysin graafiseen käyttöliittymään [29].
Teksti- tai merkkipohjaisen käyttöliittymän näytössä olevia pikseleitä ei ohjata yksitellen, vaan kaikki siinä suoritettavat komennot ja toiminnot, joita näytölle tulostetaan, tuodaan esitettäväksi käyttämällä näytönohjaimessa olevaa merkkigeneraattoria. Merkkigeneraattorissa on yleensä sisäänrakennettuna ASCII tai ANSI-standardin mukaiset merkkikokoelmat, joita käytetään merkkien tulostamisessa. Näytön tiloina käytetään yleensä VGA:ta tai sitä vanhempia näyttötiloja, kuten EGA:ta ja CGA:ta. Tekstipohjainen käyttöliittymä eroaa komentopohjaisesta käyttöliittymästä siten, että tekstipohjaisessa käyttöliittymässä voidaan käyttää koko näytön tilaa hyödyksi tiedon esittämisessä. Näin ollen komentopohjaisesta käyttöliittymästä tuttu riviltä riville tulostus ei ole enää välttämätöntä. Lisäksi tekstipohjaisessa käyttöliittymässä on jo mahdollista toteuttaa hiiren ohjaus, mikä mahdollistaa hiiren käyttämisen. Tyypillisin esimerkki merkkipoh-

jaisesta vielä nykyisinkin melko laajassa käytössä olevasta ohjelmistosta on BIOS. BIOS:lla voidaan ohjata tietokoneen päätoimintoja, kuten muistin käyttöä, kellotaajuuksia, yms. [29].

3.1.3 Graafinen käyttöliittymä

Maailman ensimmäinen Alto tietokone valmistettiin Xerox PARC:ssa vuonna 1973. Tämä tietokone oli ensimmäinen, jossa oli kaikki mahdolliset ulkoiset ja sisäiset komponentit ja elementit modernin graafisen käyttöliittymän käyttöönoton mahdollistamiseen. Näitä ulkoisia ominaisuuksia oli muun muassa hiiri, bittikarttanäyttö ja graafisten ikkunoiden näyttämisen mahdollisuus. Vuonna 1981 ilmestyi Alto tietokoneesta uusi paranneltu versio, jota monet pitävät maailman ensimmäisenä todellisella graafisella käyttöliittymällä varustettuna tietokoneena. Muutama vuosi Alton julkaisemisen jälkeen, vuonna 1984 Apple julkaisi oman Macintosh tietokoneensa, joka sisälsi Applen oman graafisen käyttöliittymän. Tätä aikaa voidaankin sanoa lopulliseksi graafisten käyttöliittymien ja hiirien syntymisen ja markkinoille siirtymisen ajanjaksoksi. Graafisten käyttöliittymien helppokäyttöisyys ja käyttäjän suorittamien erilaisten toimintojen väheneminen edesauttoivat käyttöliittymien yleistymistä ja kehittymistä hyvinkin nopeassa tahdissa [30].
Graafista käyttöliittymää (graphical user interface, GUI) voidaan kuvata sellaiseksi käyttöliittymäksi, jossa käyttäjän vuorovaikutus sovelluksen kanssa tapahtuu suoravaikuttamisen periaatetta noudattamalla. Usein graafinen käyttöliittymä koostuukin erilaisista ikkunoista, valikoista, valintapainikkeista, teksteistä, kuvista ja muista graafisista elementeistä [31].

3.2 Käyttöliittymän suunnittelu

Seuraavassa paneudutaan käyttöliittymiin ohjelmistoissa. Samalla pyritään pureutumaan siihen, mitä käyttöliittymäsuunnittelu on, mitkä ovat käyttöliittymäsuunnittelun perusteita ja periaatteita. Lisäksi käsitellään, miten käyttöliittymän suunnitteluprosessi yleensä etenee ja mitä tarkoitetaan käytettävyydellä, kun puhutaan käyttöliittymistä. Lisäksi pohditaan, mitä kannattaa ottaa huomioon, kun suunnitellaan käyttöliittymiä käyttäjälähtöisesti paremman käyttökokemuksen saavuttamiseksi.

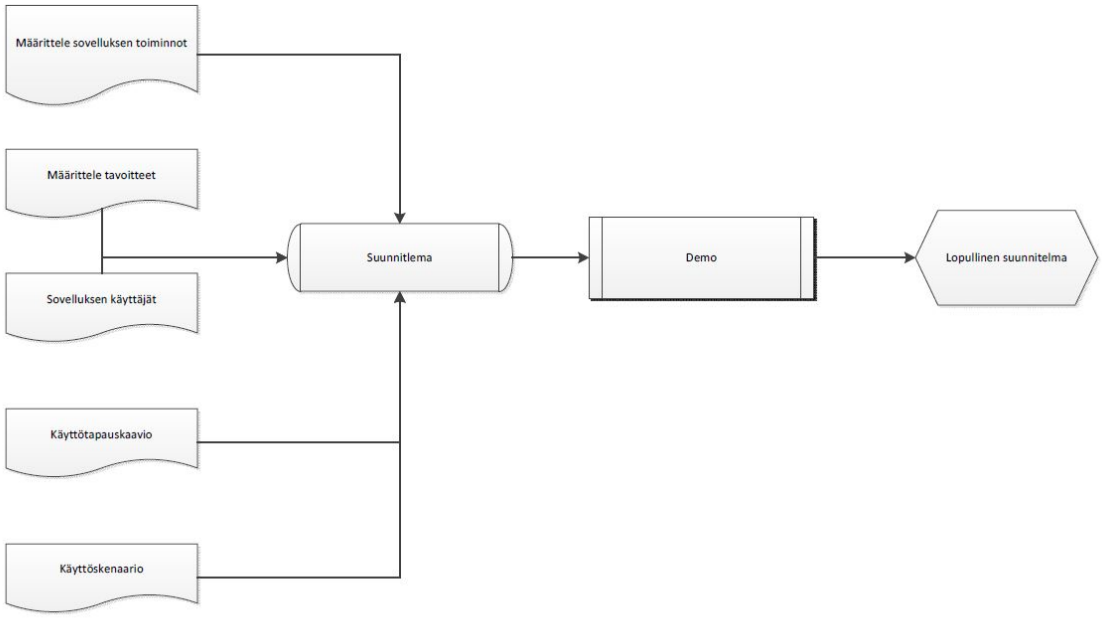
3.2.1 Käyttöliittymäsuunnittelun perusteita ja periaatteita

Ohjelmistokehityksessä, -suunnittelussa ja käyttöliittymäsuunnittelussa lähtökohtana ovat lähes aina ohjelmistoa käyttävät käyttäjät ja heidän tarpeitaan palvelevat toiminnot. Näin ollen ohjelmiston käyttäjät olisi hyvä identifioida mahdollisimman hyvin ja heidän tarpeensa kartoittaa, jotta toteutettava käyttöliittymä tai koko ohjelmisto vastaisi mahdollisimman hyvin käyttäjien tarpeita. Suunnittelijan pitää ottaa huomioon, että ohjelmaa ei tehdä omaan käyttöön, vaan ohjelmistoa tai käyttöliittymää suunnitellaan erilaisille ja eritasoisille käyttäjille. Sovelluksen tulee tukea mahdollisimman hyvin käyttäjän tarpeita, tavoitteita ja toimintatapoja tietyn yksilöllisen toiminnon suorittamiseksi. Käyttäjän kannalta katsottuna toiminnon suorittamisen pitää olla toteutettu mahdollisimman suotuisalla ja yksinkertaisella tavalla sekä asianmukaisilla ja kohtuullisilla vaiheilla [32].

Käyttöliittymäsuunnittelussa pääperiaatteena pitää olla käyttäjän toiminta ja hänen suorittamansa operaatiot sovelluksen kanssa. Lisäksi pitäisi pyrkiä ottamaan huomioon se, että käyttökokemuksesta saataisi käyttäjälle mahdollisimman sujuva ja miellyttävä kokemus, jolloin käyttäjä kokee hyötyvänsä järjestelmän käyttämisestä, eikä näe sitä suurena pakollisena rasitteena. Rasitteena kokeminen saattaa vaikuttaa ohjelmistoon ja käyttöliittymään negatiivisesti. Tämän takia käyttäjän toiminnan ja toimintaympäristön ymmärtäminen on ensimmäinen askel, jota käyttöliittymäsuunnittelussa pitää ottaa huomioon. Se ei kuitenkaan ole vielä riittävä taso siihen, että voitaisi tehdä erittäin hyvä ja toimiva käyttöliittymä. Pitää myös tutustua käyttäjien nykyiseen tapaan tehdä yksittäiset tehtävät ja saada tietää toiminnallinen syy tehtävään. Tämän jälkeen pitää ymmärtää toiminnan todellinen syy sekä mikä on todellinen tarve tehdä käyttöliittymästä sellainen, joka tukee tuota todellista tarvetta. Lisäksi tulee pohtia onko mahdollista tehdä joitakin asioita toisin ja tehokkaammin käyttöliittymän avulla. Esimerkiksi ohjekirjaa lukiessa on monesti tarve selata sivuja nopeasti, tällaisissa tilanteissa sivujen käsittelyä nopeuttaa hyvä ja kattava sisällysluettelo. Kun ohjekirjasta tehdään online-versio, ei ole tarvetta nopeaan sivujen kääntelyyn, vaan parempi vaihtoehto tähän on tehokas hakutoiminto, jolla asia voidaan hakea nopeasti näytettäväksi [32].

3.2.2 Suunnitteluprosessi

Suunnitteluprosessi voidaan jakaa erilaisiin vaiheisiin, joita voidaan myös täydentää erilaisilla suunnittelutiedoilla. Vaiheita voivat olla muun muassa sovelluksen eri toimintojen määritteleminen, tavoitteiden määrittäminen, joihin sovelluksen on tarkoitus vastata, sovelluksen yleisimpien käyttäjien tunnistaminen ja tunteminen, sovellusta kuvaavien käyttötapauskaavioiden ja käyttöskenaarioiden tekeminen, demojen tekeminen käyttöliittymästä sekä lopullisen suunnitelman tekeminen ohjelman toteutuksesta [9]. Kuvassa 2 on esitetty prosessi kuvallisessa muodossa.



Kuva 2 Suunnitteluprosessi

Sovelluksen toimintoja määriteltäessä on suunnittelijalla pääsääntöisesti olemassa mielikuva niistä asioista ja toiminnoista, joita sovelluksen on tarkoitus tarjota käyttäjilleen. Näitä ovat muun muassa asiat, joita käyttäjät odottavat järjestelmän toteuttavan siitä saatavan informaation perusteella, jotta käyttäjän on mahdollista päästä haluamaansa tavoitteeseen. Ne voivat olla myös sellaisia toimenpiteitä, jotka pitää suorittaa järjestelmää käyttämällä, jotta odotettuihin tavoitteisiin on mahdollista päästä. Asioihin lukeutuvat myös erilaiset riippuvuudet eri toimintojen välillä, erilaisten toimintasarjojen kuvaukset sekä kuvaukset tulosten ja tulosteiden laatuvaatimuksista ja niiden käsittelystä. Edellä mainittujen lisäksi suunnittelijalla tulee olla myös näkemys siitä, kuinka järjestelmässä hoidetaan mahdolliset virhetilanteet ja sellaiset erikoistapaukset, jotka poikkeavat normaalista työnkulusta [9]. Tällaisia poikkeustapauksia voi olla esimerkiksi kassa järjestelmässä tapahtuvan normaalin ostotapahtuman peruuttaminen tai tuotteen hinnan muuttaminen käsin sen normaalin lukemisen jälkeen.

Käyttöliittymän suunnittelijan on hyvä myös tuntea tyypillinen ohjelmiston käyttäjä, sillä ohjelmistoa tehdään käyttäjän kaltaiselle ihmiselle, eikä kehittäjälle, testaajalle tai muulle ”ammattilaiselle”. Näin ollen on otettava huomioon se, että tavallinen käyttäjä ei välttämättä ole tietokone-ekspertti, vaan saattaa olla vasta tietokoneen käyttöä aloitteleva henkilö. Käyttäjä ei välttämättä pysty erottamaan laitteiston ja ohjelmiston ongelmia toisistaan, vaan virheen sattuessa hänellä on edessään vain käytön estävä tai käyttöä rajoittava ongelma. Tämä ongelma saattaa estää häntä tekemästä haluamaansa toimintoa

loppuun. Näin ollen käyttäjistä on hyvä tehdä kuvaukset, jotka vastaavat ainakin seuraavassa kuvattuihin kysymyksiin, millaisia peruskäyttäjät ovat, mitä he arvostavat ja millaisia piirteitä heidän käyttäytymisessään voidaan havaita. Lisäksi on hyvä pyrkiä löytämään vastaukset kysymyksiin, miksi joku haluaa käyttää sovellustasi, mitä hyötyä käyttäjälle on sovelluksesta sekä mitä lisäarvoa tai hyötyä käyttäjä saa sovelluksen käyttämisestä [9].

Käyttöliittymän suunnittelijan on myös hyvä tehdä käyttöliittymästä/sovelluksesta käyttötapaukset ja käyttöskenaariot. Niitä tehdessä on hyvä pitää mielessä myös käyttäjän käyttökokemus ja mahdollisten osaamistasojen kirjo, koska käyttötapaukset ja käyttöskenaariot usein auttavat hahmottamaan käyttäjän toimintaa sovelluksen/palvelun kanssa. Myös käyttäjän saattaa olla huomattavasti helpompi hahmottaa sovelluksen toimintaa käyttötapausten avulla. Käyttötapaus kuvaa myös, mihin reaalimaailman tapahtumaan sovellusta/palvelua tarvitaan ja mihin sitä mahdollisesti voidaan käyttää. Käyttötapaukset kuvataan yleensä sanallisesti ja niissä voidaan kertoa, esimerkiksi sovelluksen nimi, käyttötapauksen nimi, suorittajat, käytettävyysvaatimukset, esiehdot, käyttötapauksen varsinainen kuvaus, poikkeukset sekä lopputulos. Kuvassa 3 on esimerkki käyttötapaus kaaviosta ja sen alapuolella kuvassa 4 on käyttötapauksen sanallinen kuvaus.

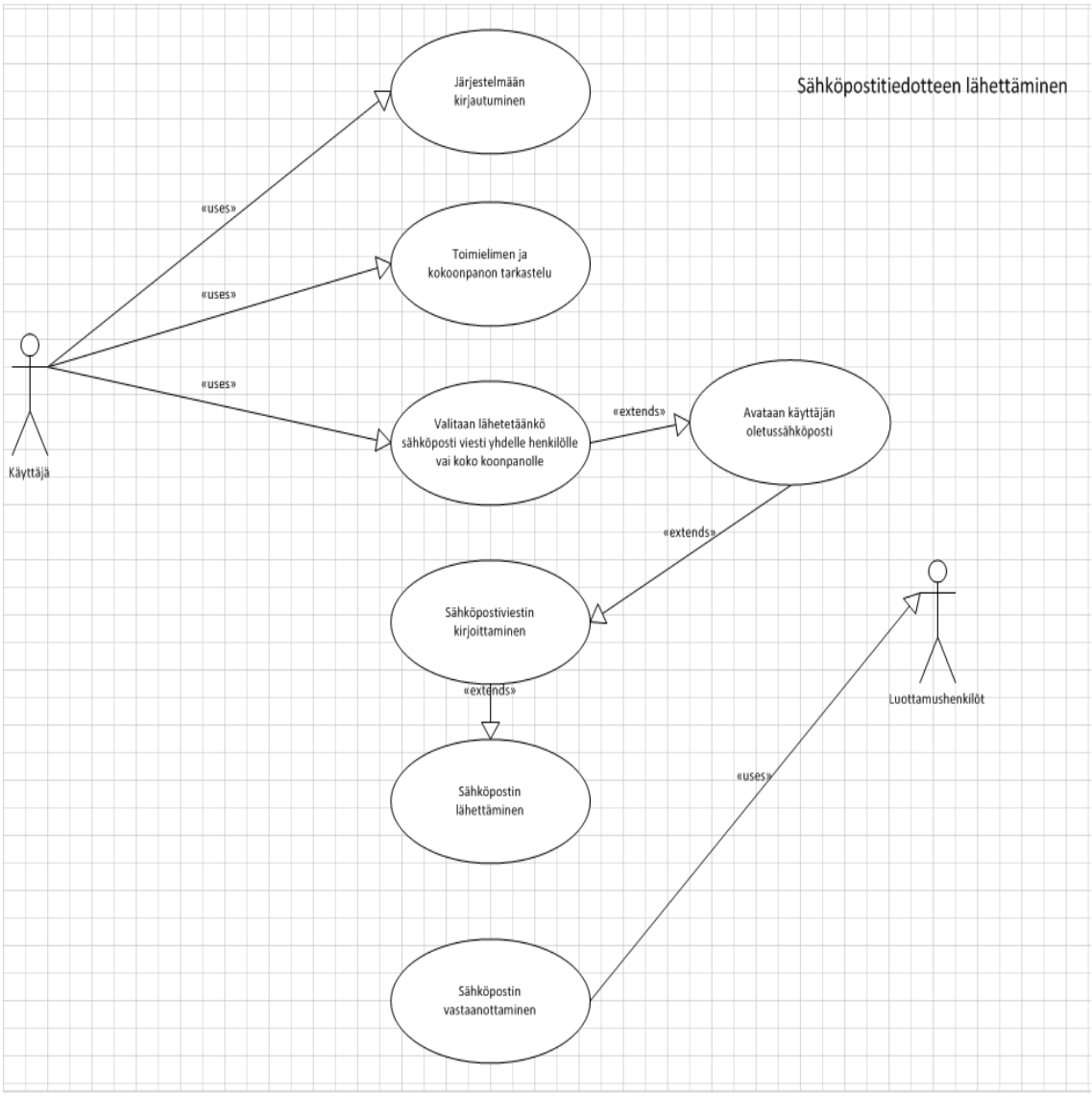
Käyttöskenaarioissa taas puolestaan kuvataan sovelluksen peräkkäiset tapahtumat, joita tarvitaan tehtävän suorittamiseen. Toisin sanoen se kuvaa yksittäisen tapahtuman/polun, jota seurataan tehtävän suorittamiseksi. Normaalisti käyttöskenaarioita tehdään useita ja niissä pyritään kuvaamaan mahdollisimman monipuolisesti perustoimintoja, epätodennäköisiä toimintoja sekä erikoistapauksia. Näin ollen voidaankin sanoa, että käyttöskenaarioiden avulla opitaan tuntemaan käyttäjiä ja saadaan selville, mitä käyttäjän ongelmaa sovelluksella pyritään ratkaisemaan [9].

Käyttöliittymäehdotelma pyrkii tukemaan ja kuvaamaan tarkemmalla tasolla käyttötapauksia ja käyttöskenaarioita. Niissä pyritään kuvaamaan mahdollisimman tarkasti yhden tai useamman näytön tapahtumat, riippuen käyttötapausten ja käyttöskenaarioiden esittämistä toiminnoista. Usein toiminnot tulee olla helposti saatavilla ja ymmärrettävissä. Kun käyttötapaukset, käyttöskenaariot ja alustavat käyttöliittymäkomponentit tai osiot on arvioitu ja testattu, voidaan tehdä lopullinen suunnitelma käyttöliittymästä ja sen tarjoamista toiminnoista [9].

3.3 Johdatus käytettävyyteen

Tässä luvussa käydään läpi käytettävyyden historiaa ohjelmiston kehityksessä. Tarkastellaan, mikä on johtanut siihen, että käytettävyyteen ohjelmistoissa on pitänyt ruveta kiinnittämään huomiota. Lisäksi kerrotaan mitä tarkoitetaan käytettävyydellä ja miten ISO standardointijärjestö määrittää käytettävyyden, sekä millaisista osakokonaisuuksista käytettävyys muodostuu ja mikä yleensä mielletään käytettävyydeksi. Lisäksi tarkas-

tellaan, mitä käytettävyys tieteenalana tarkoittaa ja mitä tieteenalalla tehdään. Luvun lopuksi tarkastellaan mitä tarkoittaa, kun puhutaan käyttöliittymien käytettävyydestä ja mistä muodostuu käyttöliittymän käytettävyys, sekä mitä tarkoitetaan käytettävyyden mittaamisella käyttöliittymissä ja mistä muodostuu hyvän käytettävyyden omaava käyttöliittymä.



Kuva 3 Käyttötapauskaavio sähköpostitiedotteen lähettämisestä

Sähköpostitiedotteen lähettäminen

Jokaisen henkilön, joka haluaa lähettää järjestelmän avulla sähköpostia tietylle kokoonpanolle tai henkilölle pitää

1. Kirjautua järjestelmään
2. Etsiä haluttu kokoonpano/henkilö
3. Painaa lähetä – painiketta, jolloin avataan käyttäjän koneen oletussähköposti, jossa sähköpostin muokkaus ja lähetys tapahtuu.

Vastaanottaja tai vastaanottajat saavat sähköpostin käyttäjän sähköpostiosoitteesta.

Kuva 4 Käyttötapauksen sanallinen kuvaus

3.3.1 Käytettävyyden historiaa ohjelmistokehityksessä

1970 -luvulla ohjelmistokehityksessä yleistyivät ohjelmistot, jotka olivat suorassa vuorovaikutuksessa käyttäjän kanssa. Näin käyttöliittymäsuunnittelusta tuli yksi tärkeimmistä osa-alueista ohjelmistotuotannon komponenteista ja siihen myös panostettiin entistä enemmän. Mitä enemmän niihin panostettiin ja ohjelmistot kehittyivät, sen interaktiivisemmaksi käyttöliittymät muuttuivat. Samalla käyttöliittymäsuunnittelu tuli kokoajan tärkeämmäksi ja ohjelmistokehityksen tuli ottaa huomioon kehityksessä ohjelmistojen käyttäjät ja sitä kautta ohjelmistojen käytettävyys ja käytön sujuvuus. Vielä 1960 -luvulla ohjelmistojen käyttäjät olivat pääosin ohjelmistoalan ammattilaisia, jotka kehittivät ja hallinnoivat ohjelmistoja organisaatioissa. Kuitenkin tietotekniikan laitteistojen ja ohjelmistojen saatavuuden parantuminen, hintojen alentuminen ja siitä seurannut laitteiden yleistyminen johti siihen, että käyttäjäkunta muuttui. Käyttäjiä oli huomattavasti enemmän ja siten myös eritasoisia käyttäjiä. Käyttäjäkunnan kasvaessa ja muuttuessa ohjelmiston asennusta, muokkaamista ja käyttöä oli helpotettava huomattavasti totutusta. Tämä johti siihen, että asiantuntijat eivät olleetkaan enää tyypillisiä käyttäjiä, vaan eritasoiset käyttäjät asensivat ja hallinnoivat omia ohjelmistojaan tietokoneissaan. Varsinkin interaktiivisissa järjestelmissä piti ottaa huomioon entistä enemmän ohjelmiston käytettävyyttä. Käytettävyydellä tarkoitetaan yleensä ohjelmiston käytön helppoa oppimista, helppokäyttöisyyttä ja käyttäjä tyytyväisyyttä [33].

3.3.2 Käytettävyys

Osana tuotetta ja sen ominaisuutena käytettävyyden on tarkoitus kuvata ohjelmiston toimintojen sujuvuutta, kun niitä käytettään. Toisin sanoen miten helposti käyttäjän on mahdollista päästä haluamaansa lopputulokseen tuotteen toimintoja käyttämällä. Voidaankin sanoa, että käytettävyydestä puhuttaessa tarkoitetaan pääasiassa ihmisen ja koneen keskinäistä vuorovaikutusta operaatiota suoritettaessa [34].
Käytettävyydellä määritellään usein myös kyseessä olevan ohjelmiston tai järjestelmän toimintojen käytön helppoutta, mukavuutta ja toimivuutta siinä tarkoituksessa, johon se on kehitetty. Ohjelmiston käytettävyys on samanaikaisesti itsenäinen ja riippuvainen tarkastelunkohde. Se kytkeytyy syvällisesti ohjelmiston tarjoamiin toimintoihin ja ominaisuuksiin. Ohjelmiston toiminnoilla pyritään kertomaan, kuinka hyödyllinen ohjelmisto on omassa käyttöympäristössään. Vastaavasti käytettävyys pyrkii kertomaan sen, kuinka onnistunutta käyttöliittymien tarjoamien toimintojen käyttäminen on [35]. Käytettävyyteen vaikuttaa myös oleellisesti ohjelmistosta löytyvien virheiden määrä. Jos virheitä on ohjelmistossa paljon, eikä käyttäjä voi luottaa ohjelmistoon, ei se myöskään anna käyttäjälle sujuvaa ja hyvää käyttökokemusta.
Käytettävyydestä voidaan sanoa, että se koostuu erilaisista osakokonaisuuksista. Näitä ovat muun muassa opittavuus, muistettavuus, tehokkuus, pieni virhealttius ja miellyttävyys. Usein käytettävyyden ja käyttöliittymien yhteydessä puhutaankin siitä, miten intuitiivista on käyttää käyttöliittymää. Intuitiivisuudella tarkoitetaan sitä, miten tuttua käyttäminen on käyttäjälle. Tuttuuteen vaikuttaa hyvin paljon käyttäjän aikaisemmat käyttökokemukset [34].
Käytettävyyden tieteenala käsittelee ja tutkii ominaisuuksia, joilla pyritään tekemään tuotteen käytettävyydestä hyvää tai huonoa. Käytettävyys koskee sellaisia menetelmiä, joilla voidaan suunnitella käytettävyydeltään hyviä tuotteitta. Lisäksi se käsittelee sellaisia menetelmiä, joilla valmiin tuotteen käytettävyyttä voidaan arvioida. Myös suunnittelumenetelmiä tukevat menetelmät, kuten käyttäjän mallintaminen ovat keskeisessä roolissa. Käyttäjän mallintamisesta saatavan tiedon avulla pystytään luomaan ja suunnittelemaan käytettävyydeltään parempia käyttöliittymiä [34].
Kansainvälisen standardointijärjestön, ISO:n, määritelmä käytettävyydestä kuvaa, miten hyvin käyttäjät pystyvät käyttämään käytössä olevia työvälineitä toimintojen suorittamiseen, tietynlaisessa ympäristössä päästäkseen asetettuihin tavoitteisiin. ISO:n standardissa, ISO 9241, käytettävyyden tarkastelun kohteiksi määritellään käyttäjä, hänen tehtävänsä, työvälineensä ja toimintaympäristönsä [34].
Voidaan sanoa, että ohjelmisto on käyttökelpoinen loppukäyttäjälle vasta silloin, kun se on käytettävä. Pelkästään helppokäyttöisyys ei siis ole riittävä peruste käytettävyydelle. On myös tärkeää ottaa huomioon, että käytettävyys voi riippua monista muista asioista, kuin käyttöliittymästä. Näitä ovat muun muassa ohjelmiston käyttöohjeistus, ohjelmiston käyttöympäristö ja käytön tarpeellisuus, sekä jo aikaisemmin mainittu ohjelman yleinen toimivuus.

3.3.3 Käytettävyys käyttöliittymässä

Kaikki ihmisen käyttämät välineet sisältävät jonkinlaisen käyttöliittymän. Esimerkiksi lapiossa on käyttöliittymä, siinä on varsi ja lapio-osa, joilla on molemmilla vaikusta tuotteen käytettävyyteen. Toisena esimerkkinä kännykkä, jonka käyttöliittymä on teknisempi ja monipuolisempi kuin lapion. Voisi sanoa, että kännykässä on fyysinen ja ohjelmistollinen käyttöliittymä yhdistettynä, kun taas lapiossa on vain fyysinen käyttöliittymä. Käyttöliittymä on yleensä se osa, mikä on näkyvä ja kosketeltavissa oleva osa laitetta ja käytettävyyttä. Suurin osa ohjelmistokehittäjän huomiosta ja käyttäjän kritiikistä onkin kohdistunut juuri käyttöliittymään ja sen käytettävyyteen. Käyttöliittymällä tarkoitetaan usein ohjelmiston nähtäviä, kuultavia ja mahdollisesti myös kosketeltavia osia. Käytettävyys on kuitenkin myös muuta kuin ohjelmiston näkyvät osat, sillä käytettävyyteen kuuluu myös ohjelmiston näkymättömiä osia kuten ohjelmiston käytön opittavuus, käyttäjälle muodostuva sisäinen logiikka, subjektiivista tyytyväisyyttä sekä ohjelmiston vasteaikoja, jotka edesauttavat käyttäjätyytyväisyydessä ja käyttökokemuksessa. Näin ollen käytettävyyden mittaaminen ei siis ole ainoastaan käyttöliittymän arvioimista vaan käyttäjän koko käyttökokemuksen huomioimista [36].

Käytettävyyden mittaamisella ei siis tarkoiteta ainoastaan käyttöliittymän toimintojen, visuaalisuuden ja kokonaisuuden arvioimista, vaan käyttäjän käyttökokemuksen kokonaiskokemuksen arvioimista ohjelmiston omassa käyttöympäristössä. Siihen saatetaankin usein liittää virheellisesti sosiaalinen hyväksyttävyys, mikä on yleensä rajattu perinteisen käytettävyyden ulkopuolelle [36].

Käyttöliittymän tulisi toimia samalla tavalla mahdollisimman loogisesti koko sovelluksen läpi. Käyttäjän siirtyessä aivan uuteen käyttöliittymän osioon, hän pystyy käyttämään ainakin osaa toiminnoista, koska ne ovat jo tuttuja toimintatapoja käyttäjän aikaisemman käyttökokemuksen perusteella. Tämä edellyttää, että samat toiminnot toimisivat samalla tavalla koko sovelluksessa. Epäjohdonmukaisuudet vaikeuttavat helposti ohjelmiston käyttöä ja käytön oppimista, mikä johtaa helpommin erilaisiin virhetilanteisiin ja sitä kautta käytön helppouteen ja opittavuuteen [34].

Käytettävyydeltään hyvän järjestelmän ominaisuuksina pidetään sitä, että järjestelmä on toteutettu johdonmukaisesti ja se on johdonmukaisesti käytettävissä. Lisäksi ohjelma on suhteellisen helposti opittavissa ja siinä olevat muistettavat asiat ovat helposti opittavissa ja niiden määrä on mahdollisimman pieni. Suotavaa on myös, että ohjelma reagoi käyttäjän tekemiin asioihin nopeasti ja suorittaa operaatiot siedettävässä vasteajassa. Järjestelmä soveltuu käytettävään ympäristöönsä hyvin ja se toimii virheettömästi. Järjestelmää on helppo käyttää ja järjestelmän ohjeistus ja opastus on tehty sujuvasti ja helppolukuisesti [37]. Lopuksi tärkeimpänä käytettävyyteen vaikuttavana tekijänä voidaan pitää esteettisyyttä ja käyttäjän kokema käytön miellyttävyyttä ja mukavuutta.

4 OHJELMISTON UUDISTAMINEN JA UUDISTETTAVA OHJELMISTO

Tässä luvussa käsitellään ohjelmien uudistamista ja nykyaikaistamista. Samalla kuvataan, mitä ohjelmistojen uudistaminen ja parantaminen tarkoittavat, mitä vaiheita uudistaminen kattaa, miten uudistamista tehdään sekä miten uudistamista kannattaa lähestyä. Lopuksi tarkastellaan liittyykö ohjelmiston uudistamiseen riskejä ja miten niihin voidaan varautua sekä miten niitä arvioidaan.

4.1 Ohjelmistojen uudistaminen ja nykyaikaistaminen

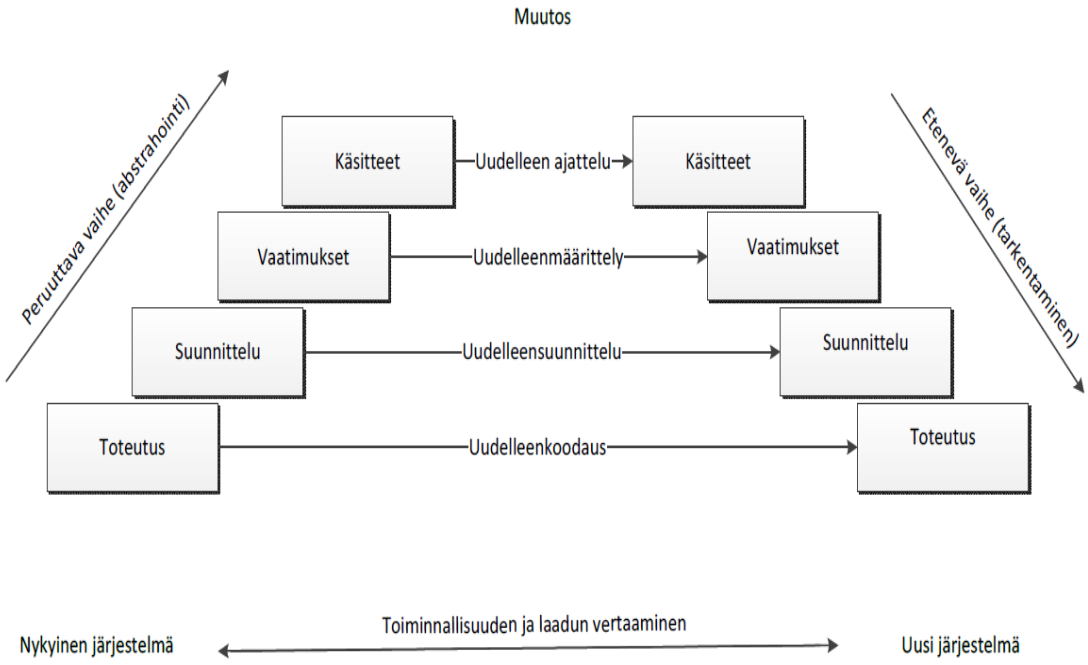
Ohjelmien uudistaminen ja parantaminen jaetaan usein kolmeen eri vaiheeseen; analyysi-, muutos- ja rakennusvaiheeseen. Analyysivaiheessa ohjelma pyritään kuvaamaan mahdollisimman korkealla abstraktiotasolla, jotta kuvaukseen perehtyvän henkilön olisi helpompi sisäistää ohjelman toimintaperiaatteet. Kuvaus pyritään tekemään mahdollisimman korkealla abstraktiotasolla myös sen takia, että se on lähempänä ihmisen omaa, luonnollista ajattelumaailmaa kuin esimerkiksi lähdekoodi. Analyysivaiheessa on tarkoituksena pyrkiä palaamaan ohjelman suunnitteluvaiheeseen. Pyrkimyksenä on saada mahdollisimman kattavasti tietoja ohjelmassa olevista suunnitteluratkaisuista. Näitä suunnitteluratkaisuita pyritään muuttamaan paremmiksi, toimivimmiksi ja mahdollisesti nykyaikaisemmiksi toiminnallisuuksiksi. Lopulta ohjelma toteutetaan uudestaan käyttäen parempia ja toimivampia suunnitteluratkaisuja [38].
Ohjelmistojen uudistaminen on läheisessä yhteydessä ehkäisevän ylläpidon kanssa. Ohjelmien uudistamiselle on olemassa erilaisia määritelmiä, joilla pyritään kuvaamaan uudistamisprosessia. Ohessa on esitettynä muutamia [38]:

- Uudistaminen on sellaista toimintaa, jonka tarkoituksena on parantaa ymmärrystä ohjelmistosta, tai minkä seurauksena ohjelmiston laatu paranee aikaisemmasta.

- Uudistaminen on sellaista toimintaa, jossa järjestelmän tai ohjelmiston mekanismeja pyritään muuttamaan siten, että ohjelmiston tai järjestelmän toiminnallisuus ei muutu vaan pysyy ennallaan.

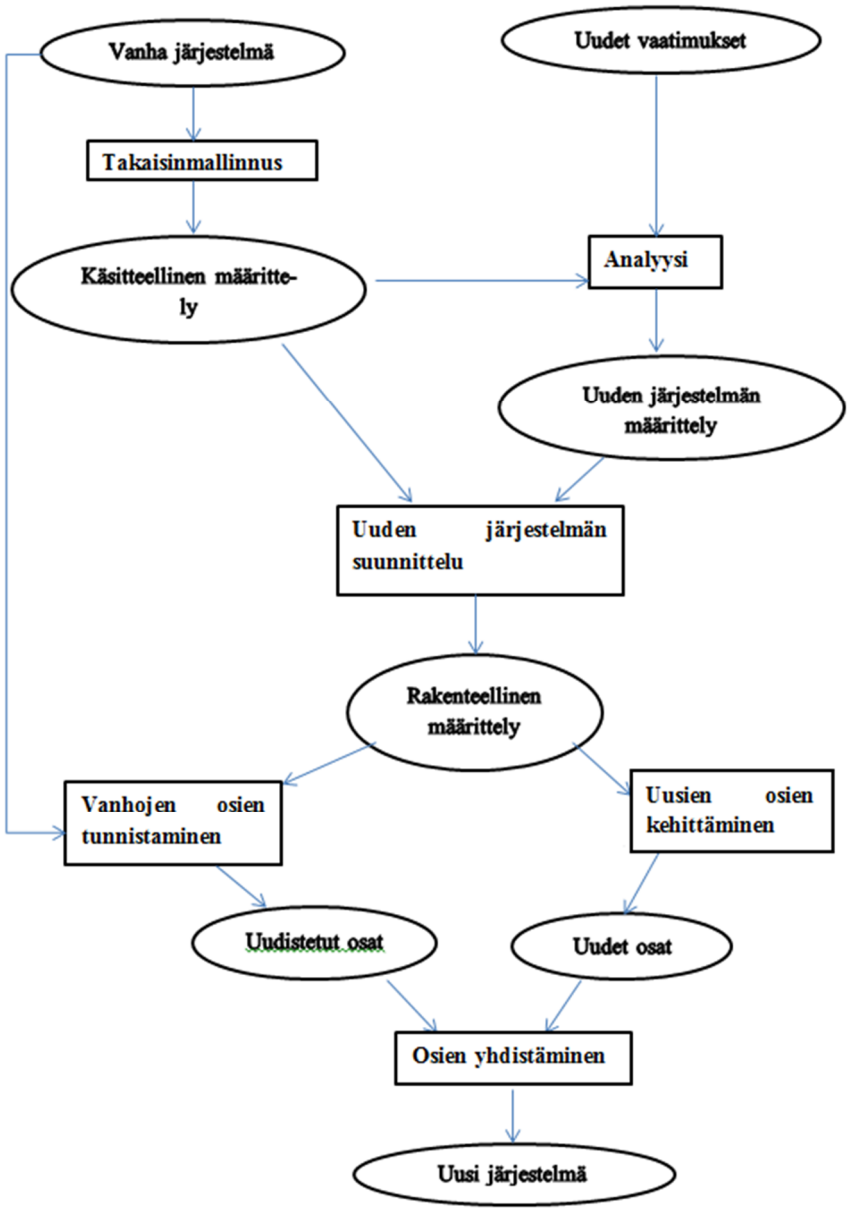
- Uudistaminen on järjestelmän tai ohjelmiston tutkimista ja muuttamista uuteen muotoon sekä tämän uuden muodon toteuttamista.

Edellä mainituista kolmesta määritelmästä voidaan pitää ensimmäistä parhaana määrityksenä, sillä se keskittyy uudistamisen päämääriin enemmän kuin menetelmiin. Uudistamalla ohjelmistoa tietyllä tavalla nämä asetetut päämäärät voidaan saavuttaa. Lisäksi ensimmäisessä kohdassa ohjelmisto käsitetään omana laajana kokonaisuutena. Näin ollen siinä on huomioitu huomattavasti laajempi näkökulma suoraan uudistamiseen kuin kahdessa muussa määritelmässä. Nämä kaksi muuta määritelmää keskittyvät pääsääntöisesti pelkän ohjelman lähdekoodin uudistamiseen [38].
Ohjelmistojen uudistamiseen kuuluu erilaisia vaiheita, joissa jokaisessa ohjelmiston abstraktiotaso todennäköisesti muuttuu jonkin verran. Yleensä korkeilla abstraktiotasoilla käsitellään sovellus- ja käyttäjäsuuntautuneita käsitteitä. Vastaavasti matalammilla tasoilla pyritään tarkastelemaan toteutuksen erilaisia yksityiskohtia. Kuvassa 5 on esitetty ohjelmistouudistamisen kolme eri vaihetta. Kuvan vaiheet ovat abstrahointi (abstraction), muutos (alteration) ja tarkentaminen (refinement). Abstrahointi vaiheessa on tarkoitus siirtyä vaiheittain ylöspäin abstraktiotasoilla eli järjestelmän yksityiskohtainen esitysmuoto on tarkoitus korvata jokaisella kierroksella abstraktimmalla esityksellä. Tällaista abstraktiotasoilla siirtymistä sanotaan takaisinmallinnukseksi. Muutosvaiheessa on tarkoitus tehdä järjestelmään muutoksia siten, että abstraktiotaso ei muutu muutosten kehittämisen aikana. Tarkentamisvaiheessa toimitaan päinvastoin kuin abstrahointivaiheessa eli tarkoituksena on siirtyä vaiheittain abstraktiotasolla alaspäin. Abstraktiotasoilla alaspäin siirtyminen vastaa yleensä ohjelmistotuotannon vaiheiden läpikäymistä, joita ovat esitutkimus, määrittely, suunnittelu, toteutus, integrointi, testaus, käyttöönotto ja ylläpito [38].
Kuvasta 5 voidaan myös todeta, että järjestelmän ominaisuuksia muutettaessa tapahtuu muutos sillä abstraktiotasolla, jolla kyseiset ominaisuudet on esitetty. Esimerkiksi, jos tarkoituksena on ainoastaan kääntää (konvertoida) ohjelma toiselle ohjelmointikielelle ei ole tarpeen tehdä takaisinmallinnusta, vaan muutos tehdään toteutustasolla. Vastaavasti mitä ylemmäksi abstraktiotasolla noustaan ylöspäin, sitä enemmän takaisinmallinnus ja tehtävien määrät kasvat [38].



Kuva 5 Ohjelmiston uudistamisen yleinen malli [**38**]

Ohjelmiston uudistaminen voi tapahtua erilaisten vaiheiden ja abstraktiotasojen kautta. Yleensä uudistaminen on tilanteesta riippuvainen, eli tarpeet määrittävät millainen abstraktiotaso ohjelmiston uudistamiselle on riittävä. Ohjelmiston uudistaminen voi tapahtua pelkästään kooditason uudelleen rakentamisella, muokkaamisella, optimoimisella tai takaisinmallinnuksen kautta, jossa noustaan myös ylemmille abstraktiotasoille. Kuvassa 6 on esitetty ohjelmiston uudistamisen eri vaiheet tilanteessa, jossa lähtökohtana on saada selville vanhan järjestelmän rakenteellinen määrittely. Kuvassa on pyritty kuvaamaan mitä eri osia järjestelmä pitää sisällään ja mitkä ovat näiden osioiden keskeiset tehtävä. Kuvasta nähdään, että vanhan järjestelmän lisäksi huomioon otetaan prosessissa järjestelmään kohdistetut uudet vaatimukset ja mahdollisesti myös kehitysideat. Uusia ohjelmaan kohdistettuja vaatimuksia analysoidaan ja analyysien pohjalta luodaan järjestelmälle uusi määrittely dokumentti. Uuden vaatimusmäärittelyn ja vanhan ohjelmiston käsitteellisen määrittelyn avulla saadaan tehtyä suunnitelma siitä, millainen järjestelmän tulee olla [38]. Näiden määritelmien pohjalta voidaan ohjelmistoa ryhtyä kokoamaan.



Kuva 6 Järjestelmän uudistaminen [38]

4.2 Uudistamiseen liittyviä mahdollisia riskejä

Ohjelman uudistamista käytetään keinona välttää riskejä tai vähentämään mahdollisten riskien toteutumismahdollisuutta ja ylläpitokustannuksia, samalla tavoitellaan sovelluksen parempaa tuottavuutta. Uudistamiseen liittyy myös erilaisia riskejä, jotka voivat realisoitua. Uudistamisen yhteydessä riskeihin kannattaa varautua etukäteen ja niistä on hyvä tehdä pienimuotoinen riskianalyysi ja kartoitukset, jotta suurilta yllätyksiltä voisi välttyä.

Yksi yleisimmistä uudistamiseen liittyvistä riskeistä on se, että uudistamista ei suunnitella riittävän huolellisesti, eikä sen toteuttamiseen varata riittävästi aikaa. Saatetaan olettaa, että kun ohjelma on jo kertaalleen toteutettu, niin ei sen uudelleen toteuttaminen voi vaikuttaa kuin toteutettuun osaan. Tämä voi johtaa tilanteeseen, jossa uudistamista tehdään muiden töiden ohessa tai sivutoimisesti. Tällöin kukaan ei oikeasti kanna vastuuta itse prosessista ja samalla unohdetaan uudistuksen ulkopuolelle jääneiden osioiden testaaminen käyttötarkoituksessaan. Tällöin voidaan päätyä myös tilanteeseen, jossa uudistamisen hyötyjä ei täysin ymmärretä. Kun ketään ei ole sitoutunut asiaan eikä asiaa hoideta asianmukaisella tavalla, siitä voi seurata myös se, ettei uudistamisesta saada kaikkia hyötyjä hyödynnettyä. Lisäksi tämä voi johtaa tilanteeseen, jossa järjestelmän uudistaminen kestää liian kauan. Pahimmassa tapauksessa järjestelmä on jo vanhanaikainen ennen, kuin se ehditään viemään markkinoille ja todelliseen käyttöön. Jos uudistamiseen ei sitouduta ja siihen ei panosteta riittävästi, jäävät uudistamiselle asetetut tavoitteet helposti hyödyntämättä [38].

Uudistamisessa esiintyvät riskit voivat liittyä myös takaisinmallinnukseen, sillä suunnitelmien ja vaatimusten tunnistaminen vanhasta ohjelmistosta voi olla työlästä ja vaikeaa. Tämä saattaa johtaa tilanteeseen, jossa koodissa oleva liiketoimintatieto jää huomaamatta ja käyttämättä tai havaitut tiedot eivät ole hyödyllisiä [38].

Riskinä uudistamisessa voidaan pitää myös tilannetta, jossa uudistamisprosessissa yritetään helposti lisätä liikaa uusia vaatimuksia tai uusia toimintoja samalla kerralla. Tietorakenteiden siirtäminen uuteen ympäristöön saattaa osoittautua erittäin hankalaksi ja se voi työllistää paljon.

Joitakin riskejä voidaan kohdistaa myös henkilöstöön ja uudistamisessa käytettäviin työkaluihin. Henkilöstö ei välttämättä ole aina sitoutunut uudistamisprojektiin tai ohjelmoijan taidot ja kokemus ovat liian vähäiset uudistamisprosessista selviytymiseen. On myös mahdollista, että työkaluista tulee prosessin kannalta niin tärkeitä, että koko projekti tulee riippuvaiseksi siinä käytetyistä työkaluista. Tämä voi johtaa siihen, että työkalut eivät toimikkaan odotetulla tavalla tai tarvittavia työkaluja ei ole saatavilla tarvittaessa [38].

5 UUDISTAMISPROSESSI

Tässä luvussa tarkastellaan ohjelmiston uudistamisprosessia käytännössä. Tavoitteena on kuvata, mikä ohjelmisto oli tässä diplomityössä uudistuksen kohteena ja mitä ohjelman uudistaminen tarkoittaa, kun sitä tarkastellaan käytännön läheisesti. Lisäksi pyritään selvittämään uudistusprosessia ja sen etenemistä käytännössä. Lopuksi käydään läpi prosessissa käytettyjä menetelmiä, jotka edesauttoivat halutun lopputuloksen saavuttamisessa.

5.1 Uudistettava ohjelmisto ja uudistamisen tarkoitus

Uudistettavana oleva ohjelmisto oli luottamushenkilörekisterinä tai mahdollisesti muuna rekisterinä käytettävä ohjelmisto. Ohjelmiston tarkoituksena on pitää kirjaa eri puolueiden kokoonpanoista ja kokoonpanojen jäsenistä, toimikausista, toimielimistä sekä luottamushenkilöistä. Myös muunlaisten henkilöistä koostuvien kokoonpanojen hallinta on ohjelman kautta mahdollista, mutta alkuperäinen käyttötarkoitus on luottamushenkilöiden tietojen hallinnan mahdollistaminen. Alkuperäinen ohjelmisto oli tehty Visual Basic-ohjelmointikielellä, joka käytti tietovarastonaan SQL-tietokantaa ja sitä ylläpidettiin SQL-serverillä. Ohjelmisto oli alkujaan tehty client -pohjaiseksi järjestelmäksi. Client -pohjaisella järjestelmällä tarkoitetaan sitä, että ohjelmisto pitää asentaa fyysisesti käytettävälle tietokoneelle, ennen kuin sitä voidaan käyttää. Näin ollen ohjelman käyttö on mahdollista vain siltä koneelta, jolle se on asennettu tai etäyhteyden välityksellä. Tietokanta voi sijaita erillisellä palvelimella ja olla kaikilla käyttäjillä yhteinen. Client -pohjaisuus voi aiheuttaa paljon työtä ylläpidollisesti, se sitoo käyttäjän mahdollisuuksia ja mahdollisesti sitoo tiettyyn paikkaan.

Ohjelmiston uudistamisprosessiin lähdettiin, koska asiakkaille haluttiin tarjota verkossa toimiva ohjelmisto, jota on mahdollista käyttää ilman erillisiä fyysisiä asennuksia. Tällöin ohjelmiston käyttö ei ole sidottu tiettyyn tai tiettyihin tietokoneisiin, joihin on asennettuna kyseen omainen ohjelmisto fyysisesti. Ylläpitoa ja huoltoa vaativissa tehtävissä voidaan palvella asiakasta paremmin ja tuote on helpommin hallittavissa, kun siitä ei ole olemassa kuin yksi instanssi yhdessä paikassa. Lisäksi tavoitteena oli, että ohjelmistoa olisi mahdollisuus käyttää miltä tietokoneelta tahansa ilman, että on sidottu johonkin ennalta määrättyyn paikkaan tai tietokoneeseen. Käytettävällä koneella pitäisi olla internetselain ja yhteys tietoverkkoon, jonne sovellus on asennettu. Yhteys voi olla muodostettu joko internetyhteyden tai lähiverkkoyhteyden välityksellä. Uudistamisen yhteydes-

sä oli tarkoituksena tehdä myös integrointeja muihin tuotteisiin, sekä tarjota rajapintoja kolmansille osapuolille, jotta rajattuja tietoja pystyttäisi jakamaan järjestelmän ulkopuolelle tai esimerkiksi organisaatioiden omille verkkosivuille. Edellä mainitut integroinnit ja rajapinnat on kuitenkin jätetty tämän työn ulkopuolelle työmäärä ja aikatauluteknisistä syistä. Uudistamisen yhteydessä käytiin läpi myös asiakkaiden esittämiä ohjelmiston kehitystoiveita, käytettävyyteen liittyviä ongelmia ja asioita. Näistä suurimpaan osaan pyrittiin vastaamaan uudistamisen yhteydessä, jotta tuotteesta saataisi paremmin asiakasta palveleva kokonaisuus ja käyttökokemuksesta sujuvampi ja kattavampi, kuin alkuperäisessä client-sovelluksessa on ollut. Näihin toiveisiin pyrittiin vastaamaan mahdollisimman hyvin, kuitenkin tuotteen palvelevuus ja tuotepohjaisuus huomioiden. Myös ohjelmiston yleiskäytettävyyteen ja ohjelmiston uusiin käyttökohteisiin laajentumiseen kiinnitettiin huomiota.

5.2 Uudistamisprosessi käytännössä

Uudistamisprosessi aloitettiin, koska uudistettavana oleva ohjelmisto nähtiin tarpeelliseksi saada Web -käyttöiseksi ohjelmistoksi. Lisäksi uudistamispäätöstä puolsi se, että haluttiin nykyaikaistaa ohjelmiston ulkoasua sekä haluttiin irtautua ohjelmiston paikallisista asennuksista (Client -pohjaisesta toteutuksesta).

Uudistamisprosessi aloitettiin tutustumalla uudistettavana olevan ohjelmiston käyttötarkoitukseen, yleisimpiin käyttöympäristöihin ja käyttöön liittyviin tarpeisiin. Tämän jälkeen siirryttiin käyttöliittymään ja toimintaan käyttämällä ohjelmistoa suljetussa testiympäristössä, jossa oli mahdollista tehdä kaikkia ohjelmiston tarjoamia toimenpiteitä vaarantamatta muuta ympäristöä. Samanaikaisesti luettiin ohjelmistosta tehtyä käyttöohjetta, jotta ohjelmiston käyttämisestä tulisi mahdollisimman tuttua ja pystyttäisiin ymmärtämään ohjelmiston käytön logiikka ja sen tarjoamat mahdollisuudet.

Ohjelman pääpiirteittäisen opettelun jälkeen siirryttiin tutustumaan ohjelmistosta tehtyyn määrittelydokumenttiin ja ohjelmiston lähdekoodiin. Määrittelydokumenttia ja lähdekoodia tutkimalla ohjelmistosta pyrittiin löytämään käytettyjä suunnittelumalleja ja lähdekoodissa olevia liiketoimintaan vaikuttavia ratkaisuja. Samalla pohdittiin ohjelmiston arkkitehtuuria ja mietittiin, miten arkkitehtuuria ja käytettyjä suunnittelumalleja olisi mahdollista tehostaa ja miten ne olisivat siirrettävissä web -käyttöiseen sovellukseen mahdollisimman tehokkaalla tavalla. Tutustuminen määrittelydokumenttiin, käyttöohjeeseen ja lähdekoodiin tapahtui käytännössä helpoiten debuggaamalla sovellusta, jolloin näkyy suoraan ohjelmiston käyttäytyminen käytön aikana ja siitä kuvatut tekniset operaatiot ja käyttäjälle kuvatut operaatiot. Näin ohjelman käyttäytymisen ja tehtyjen ratkaisuiden sisäistäminen oli huomattavasti joutuisampaa ja helpompaa. Näiden vaiheiden jälkeen tutustuttiin vielä ohjelmistoon haluttaviin uusiin toiminnallisuuksiin ja

vaatimuksiin sekä mietittiin, mitkä vaatimuksista ovat tarpeellisia toteuttaa ja miten ne ovat upotettavissa uuteen, uudistettavaan ohjelmistoon.
Prosessia jatkettiin luomalla projektisuunnitelma ohjelmiston uudistamisprosessin etenemisestä ja sovittiin uudistamisessa käytettävät ohjelmointikielet ja mahdolliset sovelluskehykset. Seuraavaksi suunniteltiin tulevalle ohjelmistolle arkkitehtuuria ja päädyttiin asiakas – palvelin mallia toteuttavaan arkkitehtuuriin. Suunnitelmien jälkeen ryhdyttiin koodaustyöhön. Ohjelma päätettiin toteuttaa kokonaan uudelleen, koska päädyimme toiseen ohjelmointikieleen ja ohjelmistokehyksen käyttämiseen.

5.3 Uudistamisessa käytetyt menetelmät

Ohjelmisto päätettiin rakentaa uudelleen toisella ohjelmointikielellä, minkä vuoksi uudistamisessa käytettiin apuna sovellettua takaisin mallinnusta.
Takaisinmallinnus voidaan yleisesti jakaa kahteen eri osa-alueeseen, joita ovat uudelleendokumentointi ja suunnitteluratkaisun jäljittäminen [38]. Projektissa takaisinmallinnuksen osalta keskityttiin suunnitteluratkaisujen jäljittämiseen. Normaalista poiketen käytettävissä oli pelkän ohjelman tukena myös määrittely- ja käyttöohjedokumentit. Suunnittelumallien ja toteutusten löytäminen oli uudelleen ohjelmoinnin kannalta tärkeää, koska sovelluksessa haluttiin säilyttää yhteensopivuus aikaisempaan versioon siten, että se on tietoja muuttamatta pystyttävä päivittämään Client -pohjaisesta toteutuksesta uuteen Web -pohjaiseen toteutukseen. Alaspäin yhteensopivuuden säilyttämiseksi tulikin kantarakenteen olla sama ja sovelluksen perustoiminnallisuuksien samantapaisia.

6 SOVELLUSKEHYS

Tässä luvussa kuvataan mikä on sovellus/ohjelmistokehys, sekä mihin sitä käytetään ja missä sitä yleisemmin käytetään. Lisäksi tarkastellaan, mitä ohjelmoijan pitää ottaa huomioon, kun käytetään sovelluskehystä. Luvun lopussa tarkastellaan voiko sovelluskehyksen käytöstä olla jotakin haittaa sen käyttäjälle.

6.1 Sovelluskehys

Sovelluskehyksen tarkoituksena on tarjota valmis toteutus käyttökohteena olevan ohjelmiston käyttämälle perusarkkitehtuurille. Sovelluskehys voi olla yksittäinen luokka, komponentti tai rajapintakokoelma, minkä tarkoituksena on toteuttaa ohjelmistojoukon yhteisen arkkitehtuurin ja niiden perustoiminnallisuuden. Sovelluskehys ei yleensä yksinään, sellaisenaan ole suorituskelpoinen kokonainen ohjelmisto, vaan se vaatii ohjelmiston ympärilleen toimiakseen. Sovelluskehys toimii jonkin ohjelmiston tai sen osan runkona, eli haluttu ohjelmisto saadaan aikaan käyttämällä ohjelmistokehystä ohjelman kehittämisen apuna. Ohjelmistokehyksen avulla voidaan toteuttaa ohjelmiston tarvitsemat osat ohjelmoimalla ne ohjelmiston kohdat uudella toiminnallisella tavalla, jossa ohjelmistokehystä on tarkoitus käyttää ohjelman tukena. Tällaista aukkojen täyttämistä kutsutaan erikoistamiseksi (specialization). Tällaisia ohjelmistojoukkoja voivat olla esimerkiksi hajautetut liiketoiminta-sovellukset tai tietyn tyyppisen graafisen käyttöliittymän omaavien sovellusten joukko [14].
Ohjelmoijan käyttäessä sovelluskehystä, hänen ei tarvitse suunnitella yksityiskohtaisesti kaikkia sovellusalueen edellyttämiä perusratkaisuja itse, vaan ne ovat useimmiten toteutettu jo sovelluskehyksen sisällä. Tavallisesti toteuttajan kannalta riittää, että ohjelmistokehittäjä tietää arkkitehtuurin toiminnallisuuden periaatteellisella tasolla. Silloin hän kykenee täyttämään tarvittavat ohjelman kohdat vaatimusten perusteella oikeanlaiseksi sovelluskehystä hyväksikäyttäen [14].

6.2 Ohjelmistokehyksen käyttökohteita

Ohjelmistokehyksiä on käytetty laajasti erilaisilla sovellusalueilla ja erilaisissa sovelluksissa. Tunnetuimpia ja käytetyimpiä sovelluskehyksiä ovat graafisen käyttöliittymän

rakentamiseen tarkoitetut sovelluskehykset, esimerkiksi Java-ympäristöön kuuluva Swing tai Googlen kehittämä GWT/SmartGWT. Käyttöliittymien rakentamiseen sovelluskehys sopii hyvin, koska interaktiivinen graafinen käyttöliittymä on luontevinta pyrkiä toteuttamaan olioperustaisesti. Lisäksi graafisella käyttöliittymällä pyritään toteuttamaan sovelluksen runko ja julkisivu. Käyttöliittymä on se ohjelman osa, mikä on suorassa interaktiossa käyttäjän kanssa. Graafisen käyttöliittymän lisäksi sovelluskehyksiä voidaan käyttää myös muissa kohteissa, kuten erilaisissa pelisovelluksissa, pankkisovelluksissa, vakuutussovelluksissa tai graafisissa editoreissa [14].

6.3 Sovelluskehyksen mahdolliset huonot puolet

Huonona puolena sovelluskehityksen käytössä voidaan mainita se, että sovelluskehykset saattavat olla hyvin laajoja kokonaisuuksia ja näin ollen niiden käyttö saattaa vaatia paljon opiskelua. Sovelluskehyksen oikeaoppinen käyttäminen vaatii kehyksen arkkitehtuurin hyvää tuntemista. Toisena huonona puolena on se, että sovelluskehys tarjoaa arkkitehtuurin lähes valmiina, jolloin hienosäätäminen on hankalaa [14]. Lisäksi sovelluskehys saattaa asettaa rajoitteita toteutukseen, koska sen avulla tuotetut komponentit eivät välttämättä taivu käyttötarkoitukseensa, eikä kaikissa tapauksissa niitä pysty itse laajentamaan. Lisäksi sovelluskehyksen tarjoamien komponenttien käyttäminen saattaa edellyttää ohjelmoijalta erilaisia ylimääräisiä toimia, kuten komponenttien alustamisoperaatioita, joita ei tarvitsisi tehdä, ilman sovelluskehystä. Tämä voi johtaa siihen, että sovelluksen koodissa koodirivien määrää kasvaa turhaan, ja näin mahdollisesti heikentää koodin luettavuutta ja pahimmassa tapauksessa ylläpidettävyyttä.

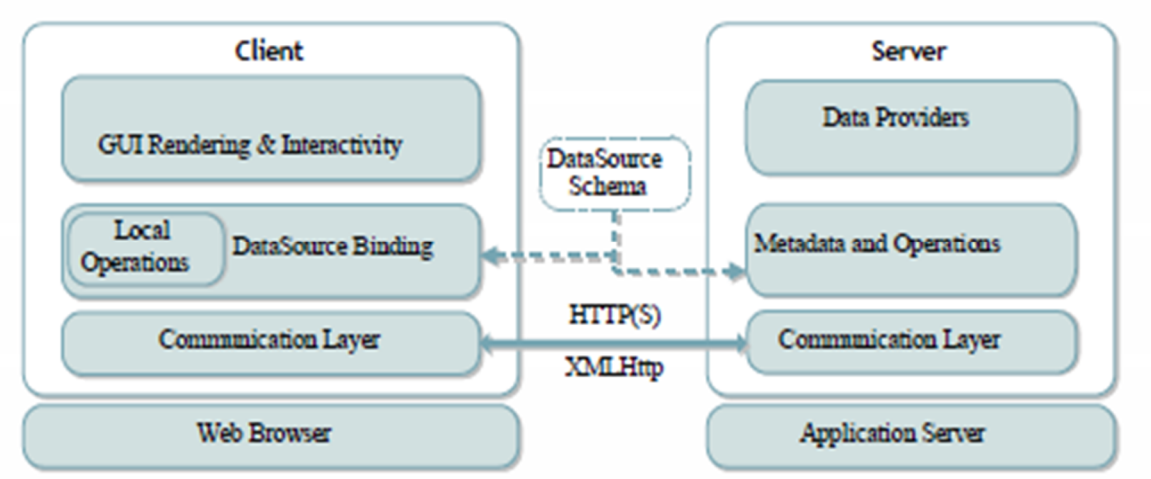
Voidaankin varmaan todeta, että sovelluskehyksen käyttäminen on aina jokin lainen kompromissi hyvien ja huonojen puolien välillä. Sovelluskehystä valittaessa tarkastellaankin usein onko sovelluskehyksen käytöstä enemmän hyötyä, kuin haittaa sillä tehtävän toiminnallisuuden tai ohjelmiston toteuttamisessa. Jos hyödyt ovat huomattavasti haittoja suuremmat ja käytöstä on huomattavaa käytännön hyötyä, päädytään yleensä käyttämään sovelluskehystä käyttökohteessaan.

7 SMARTGWT

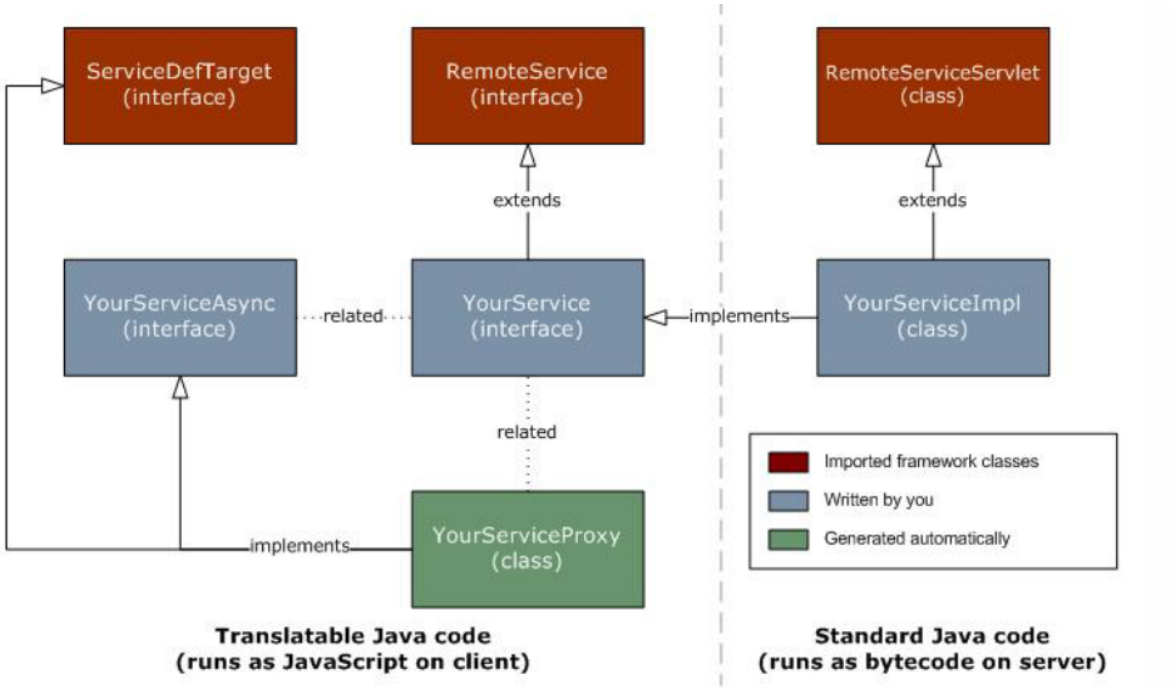
Tässä luvussa käsitellään SmartGWT sovelluskehystä, jota käytettiin ohjelmiston uudistamisen yhteydessä käyttöliittymän toteuttamisessa. Lisäksi sen avulla pystyttiin toteuttamaan clientin ja palvelimen välinen kommunikointi. Luvun tarkoituksen on kuvata lyhyesti, mikä on SmartGWT, miten se toimi, millä ohjelmointikielellä sitä käytetään ja millaista arkkitehtuuria se käyttää.

7.1 SmartGWT:n käyttämä arkkitehtuuri

SmartGWT on GWT-pohjainen sovelluskehys, jonka avulla on mahdollista tehdä RIA-(rich internet application) pohjaisia verkkosovelluksia. SmartGWT-sovelluskehys tarjoaa arkkitehtuuriratkaisun käyttöliittymäkomponenttien toteuttamisesta palvelinpuolen tapahtumakäsittelyyn asti. Sovelluskehys on tarkoitettu luomaan visuaalisia web-pohjaisia käyttöliittymiä, kohtalaisen pienellä vaivalla. Sovelluksen on tarkoitus reagoida nopealla vasteella käyttäjän tekemiin toimenpiteisiin.
SmartGWT-sovelluskehyksessä ohjelmointikielenä käytetään Java-ohjelmointikieltä. Hienosäätöä ynnä muuta voidaan tarpeen vaatiessa tehdä HTML-, CSS- ja JavaScript-kielillä. Ohjelmaa käännettäessä, automaattisen kääntäjän läpi, muodostuu asiakaspuolen (client) koodista JavaScript-koodia. Vastaavasti palvelinpuolen (server) koodi pysyy edelleen Java-koodina.
SmartGWT:n arkkitehtuuri sisältää asiakaspuolen (client) ja palvelinpuolen (server) välisen arkkitehtuurin, jonka avulla on mahdollista rakentaa RIA-tyylisiä sovelluksia, jotka kommunikoivat tieto ja palvelu kerroksilla. Kuvassa 7 on nähtävillä SmartGWT-sovelluskehyksen käyttämä arkkitehtuurikuvaus. Kuvasta voidaan nähdä, että SmartGWT tarjoaa internet-selaimelle erilaisia palveluita ja komponentteja, joiden avulla voidaan laajentaa selaimen ominaisuuksia. Vastaavasti taas palvelinpuolella se tarjoaa komponentit ja puitteet sovelluksen kehittämiselle. Komponentit voidaan ottaa käyttöön missä vain olemassa olevassa Java-pohjaisessa verkkosovelluksessa. Lisäksi asiakas- ja palvelinpuolen komponenteilla on yhteiset jaetut datalähteet (datasources), joiden tarkoituksena on kuvata liiketoimintalogiikkaa (Bussines objects) ja helpottaa toimintojen rakentamista. Esimerkiksi validointisäännöt voidaan määritellä tämän logiikan sisään, jolloin ne ovat käytettävissä SmartGWT:n tarjoamien komponenttien avulla sekä palvelin- että asiakaspuolella [13].



Kuva 7 SmartGWT arkkitehtuuri [13]



Kuva 8 SmartGWT sovelluksen kommunikointi palvelimen kanssa [39]

SmartGWT-sovelluskehyksen kommunikointi palvelimen kanssa muodostuu asynkronisilla palvelinkutsuilla. Toisin sanoen asiakaspuolen (client) luokista ei voida suoraan kutsua mitään palvelinpuolen luokkaa, vaan kutsuminen tapahtuu aina erillisten lyhyiden rajapintojen kautta. Näitä lyhyitä rajapintoja voidaan kutsua myös tynkärajapinnoiksi (stub). Edellä mainittujen rajapintojen on samalla tarkoitus toimia etäkutsuttavien metodien rajapintakuvauksina. Kuvassa 8 on esitetty kuvaus siitä, miten sovellus kommunikoi palvelimen kanssa. Palvelinpuolen luokka toteuttaa palvelinpuolella olevan tynkärajapinnan. Asiakaspuolen kutsuessa asynkronisesti palvelinpuolen metodeita, on se hyvin lähellä tavallista metodikutsua. Ainoana erona tavalliseen paikalliseen metodikutsuun on ainoastaan se, että kutsun lisäksi asetetaan kutsun jälkeen suoritettavaksi takaisinkutsumetodi (callback-method), jonka avulla voidaan todentaa selainpuolella, että kutsu on mennyt hyväksytysti läpi. Tarvittaessa sen avulla voidaan myös suorittaa haluttuja operaatioita [13].

8 UUDISTETTU OHJELMISTO

Tässä luvussa kuvataan uudistettavana oleva ohjelmisto, mitkä olivat lähtökohdat ja tavoitteet uudistamisprosessille ja sen aloittamiseen, sekä miten ohjelmiston määrittely ja suunnittelu toteutettiin. Lisäksi kuvataan mitä ohjelmistokehyksiä käytettiin. Lopuksi käydään läpi, millä ohjelmointikielellä sovellus on toteutettu ja millainen on sovelluksen arkkitehtuuri ja miten toteutus käytännössä toteutettiin.

8.1 Tavoitteet ja lähtökohdat

Ohjelmiston uudistamisen tavoitteena oli uudistaa työasemaan asennettavaa Client - pohjaista sovellusta. Uudistamisessa pyrittiin siihen, että ohjelmasta saataisi verkossa käytettävä ohjelmisto. Tämän ansiosta erillisistä asennuksista ja työasemakohtaisista ylläpitotöistä päästäisi kokonaan eroon. Uudistetun ohjelmiston pitää olla kokonainen ohjelmisto, eikä vain internet sivusto. Lisäksi uudistetun ohjelman on oltava toiminnoiltaan kattavampi ja laajempi, kuin vanha ohjelmisto ja sitä pitää pystyä käyttämään verkossa samalla laajuudella kuin vanhaa ohjelmistoa. Uudistamisen tarkoituksena oli myös nykyaikaistaa ohjelmiston ulkoasu näyttämään modernimmalta ohjelmistolta, kuitenkin säilyttäen olemassa olevan ohjelmiston käytön perusperiaatteen ja ulkoasun rakenne vanhan ohjelman tapaisena. Käyttöperiaatteeltaan ohjelmiston tuli päätoiminnoiltaan vastata vanhaa ohjelmistoa, jotta käytön tuttuus ja sitä kautta käyttökokemus ja käyttöönotto vanhalle käyttäjälle olisi jouhevampaa, ja jottei uudistetun sovelluksen käyttöönotto vaatisi isoa ja kattavaa koulutusta, vaan koulutuksessa voitaisiin keskittyä lähinnä uudistuksiin ja muuttuneisiin toimintamalleihin.

8.2 Määrittely ja suunnittelu

Ohjelmiston määrittely ja suunnittelu tehtiin käyttäen uudistettavana olevan ohjelmiston määrittelydokumenttia, josta suurin osa perustoiminnallisuuksista otettiin suoraan muokaten ne sopiviksi verkkokäyttöiseen ja usean yhtäaikaisen käyttäjän sovellukseen. Lisäksi määrittelyssä ja suunnittelussa otettiin huomioon uudet toivotut ominaisuudet ja

kehitysideat, jotka pyrittiin jo suunnittelu- ja määrittelyvaiheessa upottamaan ohjelmistoon, jotta lopputuloksesta saataisi mahdollisimman yhtenäinen ja toimiva kokonaisuus. Vanhan sovelluksen määritystä ei kuitenkaan voitu ottaa käyttöön sellaisenaan, koska vanha ohjelmisto oli tehty ja suunniteltu käytettäväksi työasemakohtaisesti, jolloin käyttäjiä on sovelluksella vain yksi kerrallaan. Kuitenkin eri sovellusten käyttäminen yhtäaikaisesti oli mahdollista, vaikka sovellukset käyttivät samaa tietokantaa. Uudistetussa verkkosovelluksessa käyttäjät käyttävät samaa tietokantaa ja sovelluksella voi olla useita yhtäaikaisia käyttäjiä samanaikaisesti. Tästä johtuen huomiota piti kiinnittää erityisesti käyttöliittymän suunnittelussa ja sovelluksen toimintalogiikan toteuttamisessa. Huomiota piti kiinnittää muun muassa ohjelmassa tehtäviin muutoksiin, jotta muutokset ovat tosiaan tekijänsä tekemiä, eivätkä ne mene ristiin toisten käyttäjän tekemien muutosten kanssa.

Myös valittu ohjelmointikieli ja sitä tukeva sovelluskehys vaikuttivat osaltaan ominaisuuksien ja määrittelyn suunnittelussa, mutta se ei kuitenkaan ohjannut toimintaa. Niiden ansiosta avautui toteutukseen uusia mahdollisuuksia, mutta samalla ne myös asettivat tiettyjä rajoitteita. Rajoitteisiin pitikin jo suunnittelu- ja määrittelyvaiheessa kiinnittää erityistä huomiota, jotta kehystä pystyttäisiin käyttämään mahdollisimman laajasti ja kattavasti, jotta siitä saataisiin kaikki sen tuomat hyödyt mahdollisimman laajasti käyttöön. Näin myös mahdollisesti vältyttäisiin haittavaikutuksilta, jos sellaisia olisi aiheutumassa. Tämä taas johti siihen, että käytettäväksi otettava ohjelmistokehys piti ennen projektia opetella hyvin, jotta sen potentiaali tunnistetaan ja tehdään asioita oikein alusta alkaen ja vältytään turhilta kommelluksilta kehystä käytettäessä. Näin vältyttiin myös ylimääräisiltä yllätyksiltä ja tiedettiin toimintatavat jo ennen projektin alkua.

8.3 Toteutuksessa käytetyt kielet, sovelluskehykset, kirjastot ja apuohjelmat

Luottamushenkilörekisteriohjelmiston uudistamisessa päädyttiin käyttämään Java-ohjelmointikieltä. Javaan päädyttiin, koska se on yrityksen muissa tuoteperheen tuotteissa käytetty ohjelmointikieli. Muita vaikuttavia tekijöitä Java-ohjelmointikielen valintaan oli muun muassa se, että Javalla toteutetut sovellukset ovat lähes alustariippumattomia. Java-kielellä toteutetut ohjelmistot toimivat PC-maailmassa Windows-, Linux ja Mac-ympäristöissä. Lisäksi Java on nimenomaan tarkoitettu erityisesti verkkopalveluiden palvelinpään toteuttamiseen ja muihin palvelimella suoritettaviin ohjelmiin.

SmartGWT-sovelluskehystä päädyttiin työssä käyttämään, koska se oli jo otettu käyttöön yrityksessä toisen tuotteen kohdalla ja kokemukset sen käytöstä olivat kohtalaisen hyviä. Lisäksi sen avulla oli mahdollista tehdä näyttäviä käyttöliittymiä suhteellisen nopeasti ja vaivattomasti. Lisäksi siinä on sisäänrakennettuna palvelimen ja Clientin väliseen kommunikointiin suunniteltu rakenne (luku 7). SmartGWT-sovelluskehyksen

avulla myös käyttöliittymän toteutuksen tuottavuus saatiin hyväksi, mikä mahdollisti erilaisten käyttöliittymäprototyyppien toteuttamisen melko pienellä vaivalla. Se taas mahdollisti paremmin käyttöliittymän käytettävyyden huomioimisen ja käyttöliittymien demoamisen. SmartGWT:n käyttöä ohjelmiston toteutuksessa puolsi myös sen tarjoamat laajat mahdollisuudet. Edellä mainittujen lisäksi SmartGWT:n käyttöä ohjelmistossa puolsivat, sillä toteutetun ohjelmiston testattavuus, johdonmukaisuus ja yhteen toimivuus muiden teknologioiden ja ohjelmistojen kanssa. Lisäksi siitä tuotetut dokumentaatiot ovat yllättävän kattavia ja kuvaavia. Hyvänä puolena voidaan myös sanoa, että käyttäjäyhteisöt ovat kohtalaisen aktiivisia. SmartGWT:n käyttö on aiheuttanut myös hankaluuksia, esimerkiksi käyttöliittymän testaaminen automaattisesti on melko hankalaa kaikilta osin. Lisäksi se pakottaa arkkitehtuurinsa takia toteuttamaan toiminnot tietyllä tavalla.

Sovelluskehyksen lisäksi ohjelmiston uudistamisessa käytettiin apuna DBPool apuohjelmakirjastoa. DBPool apuohjelmakirjaston avulla voidaan parantaa ohjelman vasteaikoja, kun käytetään tietokantoja ja tietokantayhteyksiä. Vasteaikojen parantuminen johtuu siitä, että ohjelmallisesti tietokannan käsittelyssä suurin osa ajasta menee yhteyden avaamiseen tietokantaan. DBPoolin avulla voidaan avoimia yhteyksiä käyttää, jos niitä on olemassa, kun operaatiot ovat suoritettu. DBPool hallitsee avattuja tietokantayhteyksiä, joista sovellus voi tarvittaessa ottaa yhden ja palauttaa sen takaisin säilöön, kun on sitä käyttänyt.

8.4 Uudistetun luottamushenkilörekisterin arkkitehtuuri

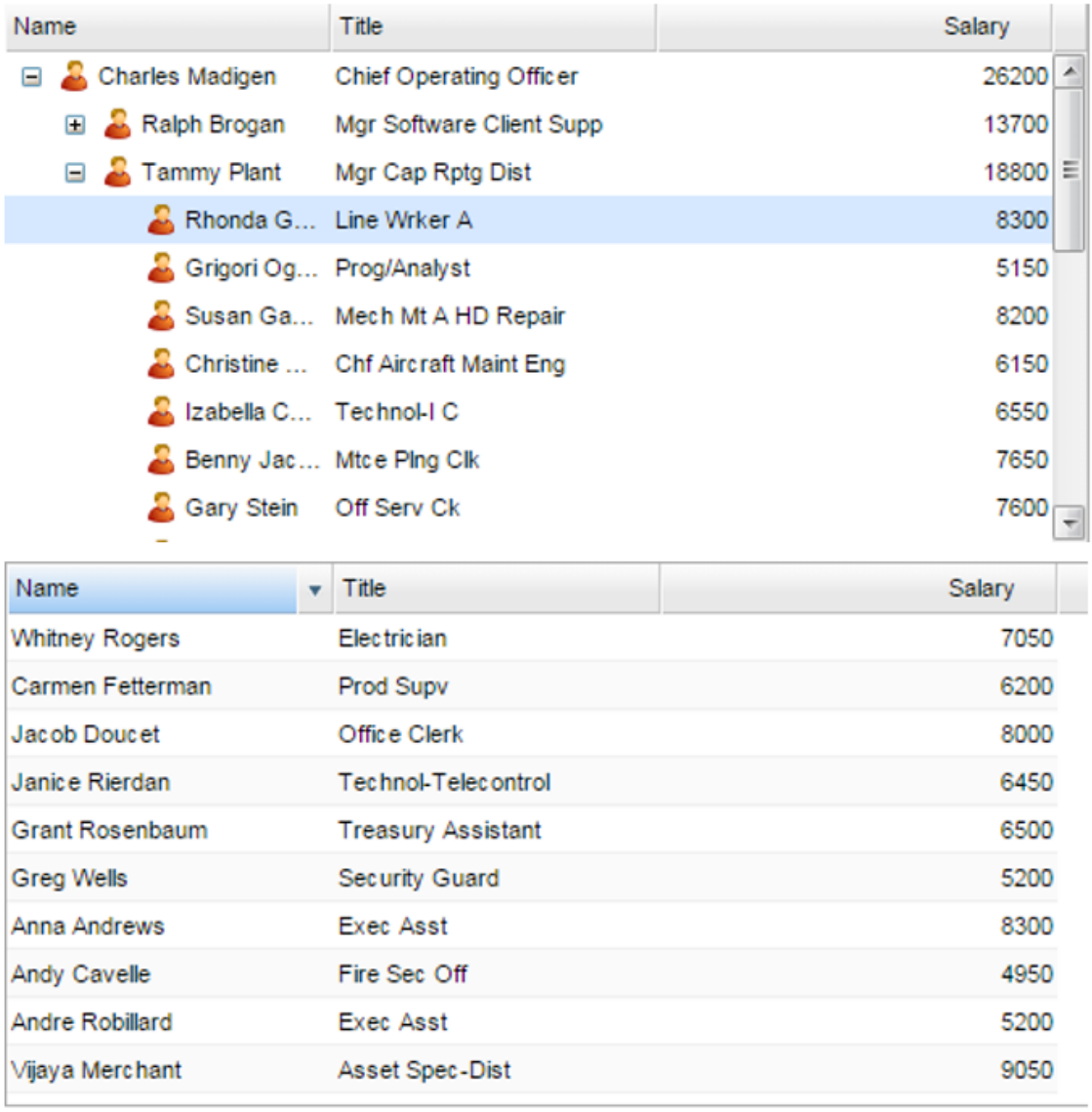
Uudistetussa luottamushenkilörekisterissä käytetty arkkitehtuuri pohjautuu siinä käytettävään SmartGWT-ohjelmistokehyksen mukaiseen arkkitehtuuriin, koska se pakottaa osittain käyttämään asiakas – palvelin arkkitehtuuria.

Asiakas-palvelin-arkkitehtuurissa sovelluslogiikka on jaettu kahteen eri osaan asiakkaiden ja palvelimien kesken. Tällä arkkitehtuurilla tehdyt sovellukset koostuvat kahdesta kerroksesta. Arkkitehtuurissa asiakkaat hoitavat tiedon esittämisen ja vastaavat käyttäjän interaktioihin ja palvelin hoitaa liiketoimintalogiikan, sekä pysyvän tiedon tallentamisen johonkin tietovarastoon.

Kyseisessä arkkitehtuurissa palvelin kapseloi hallitsemansa resurssit siten, että asiakaspään ei tarvitse huolehtia resurssien käyttöön liittyvistä teknisistä ongelmista. Palvelin pyrkii hoitamaan kaiken niihin liittyvän ja niistä huolehtimisen, jolloin asiakaspäässä niitä voidaan käyttää vapaasti. Tällaisia teknisiä ongelmia voivat olla esimerkiksi rinnakkaisuuden aiheuttamat ongelmat. Näin ollen asiakas voi pyytää palvelimelta resursseja lähes vapaasti haluamallaan tavalla.

Vastaavasti asiakaspuolella, asiakasta varten luodaan yleensä tietyn tyyppinen istunto, jonka asettamien rajojen sisällä kommunikointi palvelimen kanssa tapahtuu. Pääsääntöi-

sesti palvelinpuoli odottelee, että saa yhteydenottopyynnön asiakkaalta ja pyynnön saapuessa se suorittaa sille tarkoitetut tehtävänsä. Tehtävän suoritettuaan se vastaa asiakkaalle operaation onnistumisesta. Kun asiakas on todennut oman tehtävänsä hoidetuksi, se sulkee avaamansa istunnon [40].



Kuva 9 Uudistetussa luottamushenkilörekisteri sovelluksessa käytettyjä käyttöliittymä komponentteja [41]

8.5 Toteutus

Ohjelman toteuttaminen aloitettiin suunnittelemalla käyttöliittymä. Käyttöliittymä suunniteltiin määrittelyn ja vanhan sovelluksen avulla. Siinä haluttiin säilyttää vanhasta sovelluksesta tuttuja ja hyväksi havaittuja käytäntöjä, mutta kuitenkin ulkoasua haluttiin uudistaa ja parantaa. Ulkoasulta toivottiin, että siitä tulisi mahdollisesti informatiivisempi ja helppokäyttöisempi. Käyttöliittymän toteuttamista ja hahmottamista auttoi SmartGWT:n show case-sivusto, josta pystyi näkemään käytännössä SmartGWT-ohjelmistokehyksen tarjoamat käyttöliittymäkomponentit. Työssä käyttöliittymän pääkomponenteiksi valittiin puurakenne ja taulukkorakenne kuvaamaan päänäyttöjä. Kuvassa 9 on nähtävillä käytetty puurakenne yläpuolella ja taulukkorakenne alapuolella. Kuvat eivät ole uudistetusta luottamushenkilörekisterisovelluksesta vaan SmartGWT showcase-sivustolta.

9 OHJELMISTON TESTAUS

Tässä luvussa käydään läpi ohjelmiston testaamista. Aluksi määritellään testaus ja tarkastellaan testausta virheiden kartoituskeinona sekä tarkastellaan testauksessa havaittuja virheitä. Lisäksi mietitään, miten virheiden jakamista voisi suorittaa, sekä miten testausta voidaan jakaa eri osa-alueisiin ja mitä eri tasoja testaukseen voidaan liittää. Lopuksi kerrotaan, milloin testaus olisi riittävää, mitä dokumentteja testauksesta syntyy ja miten testaus on toteutettu tässä diplomityössä.

9.1 Mitä testaus on?

Testauksen määritelmiä on useita erilaisia. Ohessa muutama määritelmä, erään määritelmän mukaan testaus on prosessi, jossa ohjelmaa suoritetaan tarkoituksena löytää ohjelmasta virheitä, testaus on ohjelmiston laadun mittaamista, oleellinen osa testausta on siihen liittyvän dokumentaation, työkalujen ynnä muun sellaisen käyttäminen ja ylläpito, testaus on tekninen tutkimus, joka tehdään laatuun liittyvän tiedon paljastamiseksi testauksen kohteena olevasta tuotteesta, testaus on ohjelmien rikkomista [42]. Kuten edeltä nähdään määritelmiä voi olla useita eikä ne välttämättä edes tarkoita samaa asiaa. Testauksella ei suoraan voida todistaa, että ohjelmisto olisi virheetön, eikä se itsessään paranna ohjelmiston laatua. Testaus on ohjelmiston laadun mittari. Testaus ei ole ohjelman määrityksen tarkastamista. Toisenlaisen näkökulman mukaan onnistunut testin suorittaminen on sellainen, joka ilmaiseen poikkeaman ohjelman toiminnassa. Tällaisella ajattelutavalla, voidaan ohjelmasta löytää virhe, minkä löytäminen ja poistaminen parantavat ohjelman laatua. Testaajan on hyvä ajatella, että ohjelmassa on aina virheitä, jotka odottavat löytämistä [42].

9.2 Testaustyö

Testauksen työvaiheet voidaan karkeasti jakaa neljään erilliseen työvaiheeseen, testauksen suunnitteluun (testaussuunnitelma, testitapaukset), testiympäristön ja testausympäristön luontiin sekä testitulosten arviointiin ja läpikäyntiin. Edellä mainittuihin työvaiheisiin ja niihin liittyviin virheiden jäljitykseen ja korjaukseen (debugging) kuuluu pää-

sääntöisesti yli puolet ohjelmistoprojektin resursseista ja ajasta, siksi testaukseen suunnitteluun ja toteuttamiseen kannattaa todella asettaa painoarvoa jo projektin alussa. Testauksen määrä on aina jonkinlainen kompromissi ohjelmistoprojektin resurssien suhteen. Kompromissi määräytyy useimmiten ohjelmistoprojektissa käytettävän ajan, rahan, välineiden ja luotettavuuden ja siitä saavutetun varmuuden välillä [10].

9.3 Testaus virheiden kadotuskeinona

Usein testauksesta puhuttaessa tarkoitetaan melkein mitä tahansa asioiden, esineiden, toimintojen kokeilemista eri asioissa. Ohjelmistojen testauksen yhteydessä testauksella tarkoitetaan lähes aina testaamista, jossa suunnitelmallisesti etsitään virheitä kohde ohjelmistosta. Virheiden etsiminen tapahtuu suorittamalla ohjelmistoa tai ohjelmiston osiota. Erillisen laitteiston testaus määritellään yleensä suunnitelmalliseksi laitteiston käyttöympäristöön sopivuuden, toimivuuden, käytettävyyden ja kestävyyden testaukseksi [10].

Testattaessa ohjelmistoa ohjelmalle annetaan syöte, minkä johdosta ohjelmassa tapahtuu jotakin, tapahtuma johtaa toiminnon suorittamiseen tai tulokseen suorituksesta. Syötteet eivät tässä tapauksessa ole välttämättä sellaisia, jotka ovat luokiteltu ohjelman sallimiksi syötteiksi, vaan ne voivat olla mitä tahansa syötteitä, joille järjestelmä voi altistua toimintaympäristössään. Myös tulosteet voivat olla mitä tahansa järjestelmän ulkopuolelta havaittavissa olevia toimintoja [10].

Testin tulos on riippuvainen siitä, millaisia syötteitä ohjelmistolle on annettu. Lisäksi saatu tulos on riippuvainen syötteestä ja järjestelmän sisäisestä tilasta. Sisäisellä tilalla tarkoitetaan tässä tapauksessa ohjelman muuttujien arvoja, rekisterien arvoja, käskyosoittimen arvoja, levylle tallennettua tietoa sekä mahdollisesti joidenkin testattavien järjestelmien sisäisten laitteiden tilaa. Testin onnistumisen perusedellytyksenä on, että tulos on oikein ja ohjelmiston sisäinen tila on muuttunut odotetunlaiseksi. Näin ollen testaajalla on oltava hyvä käsitys ohjelman tuottamasta tuloksesta ja syöteavaruudesta. Testaajan pitää tällöin myös tietää, mikä testattavassa ohjelmassa kyseisellä syötteellä on oikeanlainen lopputulos [10].

9.4 Testauksessa havaitut häiriöt ja virheet

Haikalan ja Merijärven [10] mukaan testauksen yhteydessä sana virhe on jonkinlainen poikkeama spesifikaatiosta. Testaus ilman spesifikaatiota on mahdotonta, koska lopputuloksen oikeellisuutta ei ole mahdollista todentaa. Testauksessa käytettäviä spesifikaa-

tioita ovat toiminnallinen sekä tekninen määrittely. Testauksessa havaitun virheen vakavuus vaihtelee järjestelmän käytön kokonaan estävästä virheestä käyttäjää ärsyttäviin pieniin yksityiskohtiin.
Yleisesti arvioidaan, että ohjelmistoissa olisi ohjelmoinnin jäljiltä yksi virhe muutamaa kymmentä ohjelmariviä kohden. Lisäksi arvioidaan, että pitkään käytössä olevissa ohjelmistoissa olisi noin yksi virhe tuhatta ohjelmariviä kohden. On myös tieteellisesti todettu, että noin 5 % ohjelmistojen virheistä on sellaisia, joita ei tulla sieltä koskaan löytämään [10].

9.5 Testauksen jakaminen

Testaus voidaan jakaa useisiin erilaisiin tasoihin. V-mallin mukaisesti testaustasoja ovat moduulitestaus, integrointitestaus sekä järjestelmätestaus. Järjestelmätestausta voi seurata kenttätestaus tai hyväksymistestaus. Kuvassa 10 on esitetty testauksen V-malli. Muita testauksen yhteydessä käytettyjä termejä ovat muun muassa alfa- ja betatestaus, regressiotestaus sekä käytettävyystestaus [10].
V-mallin mukaisessa testauksessa suunnittelu tapahtuu aina testaustasoa vastaavalla suunnittelutasolla. Esimerkiksi määrittelyvaiheessa suunnitellaan järjestelmätestausta, integraatiotestaus suunnitteluvaiheessa ja moduulitestaus moduuleiden suunnittelun yhteydessä. Näin ollen myös tuloksien oikeellisuus todetaan vertaamalla testeissä saatuja tuloksia vastaavien kohtien dokumentteihin [10].

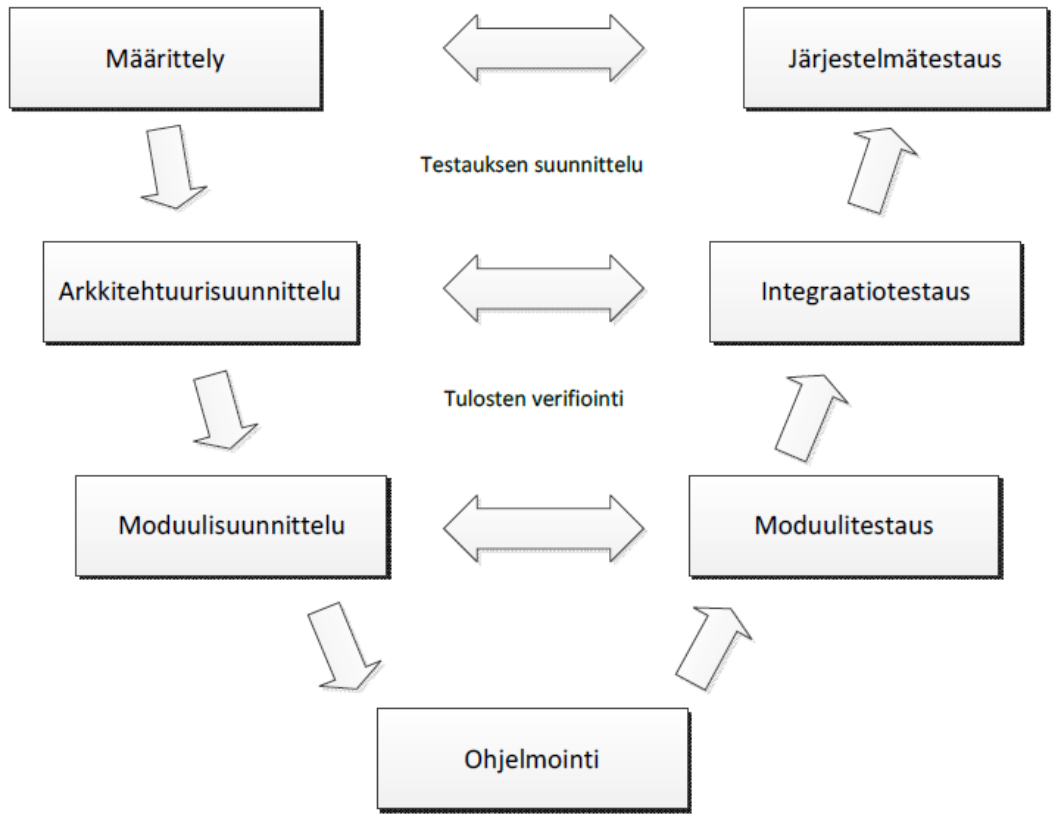
9.6 Testaustasot

Moduulitestauksessa on tarkoituksena testata yksittäistä moduulia kerrallaan. Tällöin moduulin toimintaa verrataan moduulisuunnittelun ja arkkitehtuurisuunnittelun tuloksiin. Yleensä näiden suunnitteluiden tulokset löytyvät teknisestä määrittelydokumentista [10]
Integrointitestauksella tarkoitetaan sitä, että yhdistellään moduuleita tai moduuliryhmiä (osajärjestelmiä) kokonaisuuksiksi. Painopiste pidetään kuitenkin kokoajan näiden moduulien välisten rajapintojen toimivuuden tutkimisessa. Myös tällä tasolla tuloksia verrataan yleensä tekniseen määrittelyyn, josta voidaan nähdä järjestelmän suunniteltu toiminta. Integrointitestausta tehdään yleensä rinnan moduulitestauksen kanssa, eikä sitä yleensä tarkastella erillisenä osiona moduulitestauksesta [10].
Järjestelmätestauksessa puhuttaessa tarkoitetaan yleensä koko järjestelmän testaamista, jossa tuloksia verrataan määrittelydokumentaatioon (ohjelman toiminnalliseen määritte-

lydokumentaatioon) ja asiakasdokumentaatioon. Joissakin tapauksissa järjestelmätestauksen yhteyteen voidaan liittää myös kenttätestausta tai hyväksymistestausta. Hyvän tuloksen saavuttamiseksi, järjestelmätestausta tekevien pitäisi olla mahdollisimman riippumattomia kehitystöitä, jotta tuloksista saataisi mahdollisimman totuudenmukaisia. Toiminnallisuuksien varmistamisen lisäksi järjestelmätestauksen tavoitteena on testata ohjelmiston ei-toiminnalliset osuudet [10].

Asiakkaan suorittamaa testaamista ohjelmiston toimittajan tiloissa kutsutaan yleensä Alfatestaukseksi ja vastaavasti asiakkaan itsenäistä omissa tiloissaan tapahtuvaa testausta kutsutaan Betatestaukseksi [10].

Viimeisimpänä voidaan vielä puhua käytettävyystestauksesta. Käytettävyystestauksen avulla pyritään varmistamaan, että järjestelmän käyttäjä selviää sellaisista tehtävistä ohjelmiston avulla, joihin ohjelmisto on rakennettu. Käytettävyystestaus on pääosin käyttöliittymän käytön testaamista. Käyttöliittymän käytön testausta tehdään myös ennen käytettävyystestausta käyttöliittymäprototyypin avulla [10].



Kuva 10 Testauksen V-malli [10]

9.7 Testauksen riittävyys

Haikalan ja Merijärven [10] mukaan testauksen määrän arviointi etukäteen on erittäin vaikeaa, jopa mahdotonta. Tästä johtuen testauksen lopettamiselle tulisi aina asettaa jonkinlaiset raamit ja hyväksymiskriteerit, jotka kirjataan erilliseen testausdokumentaatioon. Esimerkiksi järjestelmätestausvaiheessa testauksen riittävyyden arvioimisen kriteereiksi voidaan määritellä kumulatiivinen käyrä. Tällä tarkoitetaan sitä, kun käyrä alkaa tasaantua ja virheet alkavat vakiintua, voidaan testaus todeta riittäväksi ja lopettaa testaaminen. Moduulitestauksessa taas riittävä testauksen määrä voidaan koittaa arvioida mutkikkuusmitoilla ja virheitä kylvämällä. Tässä vaiheessa tapahtuvassa testauksessa on suurena haasteena se, että projektin vaatimat kiinteät resurssit ovat normaalisti kiinnitetty ja aikataulut on sovittu etukäteen.

Ohjelman monimutkaisuutta ja koodin rakennetta pyritään usein analysoimaan mutkikkuus- eli kompleksisuusmitoilla. Mutkikkuusmittoja voidaan käyttää muun muassa siihen, että ohjelmistosta pyritään löytämään paljon testausta vaativat moduulit, joiden tarkoitus on puolestaan auttaa määrittämään testauksen määrän suunnittelua. Kattavuusmitoilla taas yritetään varmistaa testiaineiston kattavuus testattavaa ohjelmistoa vasten. Tavoitteena on kartoittaa, että kaikki ohjelman osat tulevat kattavasti testattua. Kattavuusmitoilla voidaan myös pyrkiä todistamaan, että testausta on suoritettu riittävä määrä.

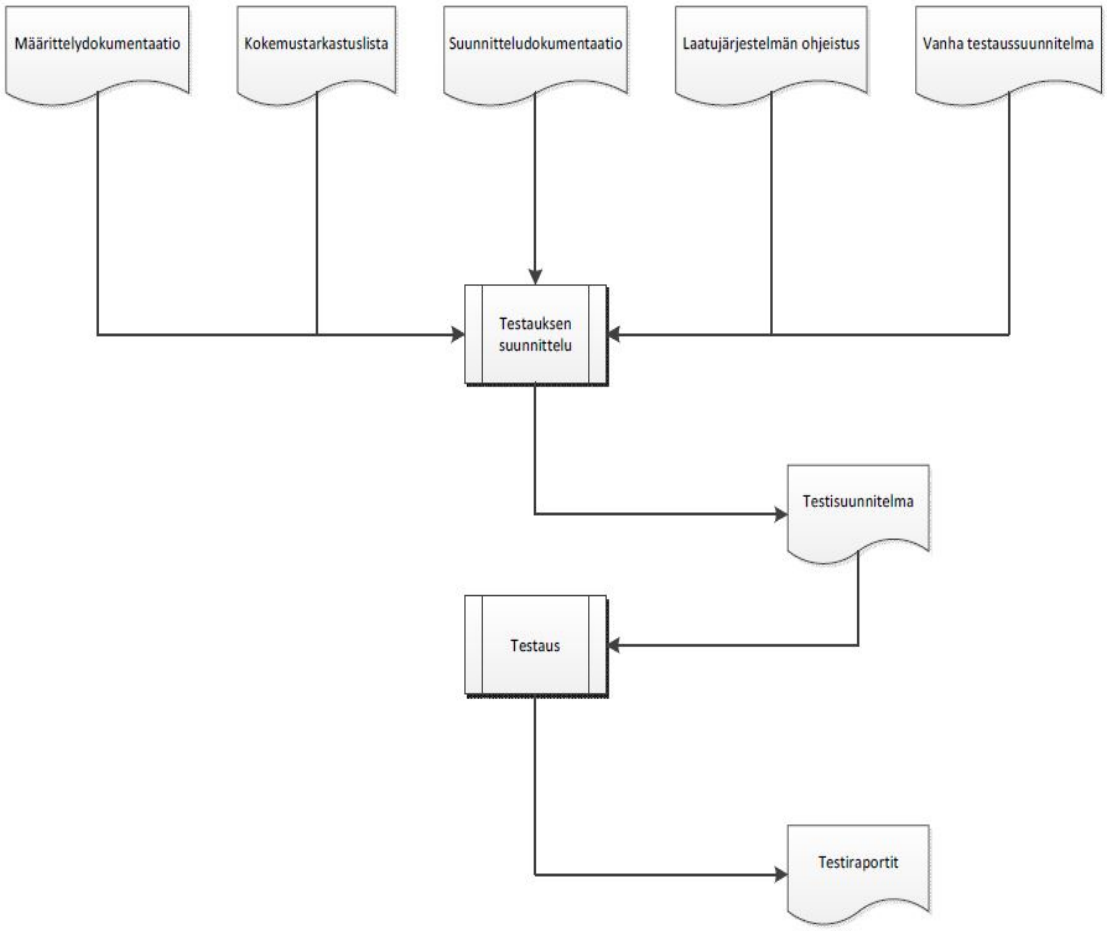
Testaus voidaan määritellä riittäväksi myös käyttäen virheiden kylvämistä. Virheiden kylväminen tarkoittaa sitä, että virheitä kylvetään tarkoituksellisesti ohjelmakoodin joukkoon ja niitä pyritään löytämään testauksen avulla. Testauksessa löydettyjen virheiden määrästä voidaan pyrkiä selvittämään ohjelmaan jääneiden virheiden määrä. Esimerkiksi oletetaan ohjelmassa olevan X oikeaa virhettä ja Y kylvettyä virhettä lisäksi oletetaan, että testauksessa on löydetty X oikeaa virhettä ja Y kylvettyä virhettä. Sitten oletetaan, että kylvetyistä ja todellisista virheistä on molemmista samassa suhteessa löydetty yhtä suuri osuus, voidaan tästä suhteesta arvioida todellisten virheiden määrä X käyttäen kaavaa X=XY/Y [10].

9.8 Testauksesta syntyvät dokumentaatiot

Testauksen yhteydessä saattaa syntyä paljon erilaisia dokumentaatioita, koska lähes jokaisesta suoritettavasta testivaiheesta tehdään jonkin asteinen testaussuunnitelma. Syntyviä dokumentteja voi olla muun muassa järjestelmätestaussuunnitelma, integrointitestaussuunnitelmat, moduulitestaussuunnitelmat sekä näiden lisäksi tehdään raportit jokaisesta kokonaan suoritetusta testistä. ISO 9000-3-ohjeistuksen mukaisesti testaussuunnitelma tulisi tehdä järjestelmä- ja integrointitestauksen yhteydessä. Ohjeistuksen mukaan moduulitestaussuunnitelman voi korvata laatukäsikirjasta löytyvällä ohjeistuk-

sella. Usein pienissä projekteissa todetaan yksi erillinen testausdokumentti riittäväksi. Tämä testausdokumentti sisältää järjestelmätestauksen, moduulitestauksen sekä integraatiotestauksen. Kuvassa 11 on esitetty, mistä komponenteista testauksen suunnittelu, testaussuunnitelma ja testaus muodostuvat. Kuvassa 12 on esitetty testaussuunnitelman rakenne sisällysluettelona. Laajempi ja perusteellisempi sisällysluettelo on mahdollista nähdä IEEE 829 standardista [10].

Hyvästä testaussuunnitelmasta tulisi selvitä, mitä testejä ollaan tekemässä, milloin, miten ja missä järjestyksessä ne tehdään, sekä mitä tuloksia kultakin testiltä voidaan ja halutaan odottaa. Testaussuunnitelmaan on tärkeää muistaa määritellä testauksen lopettamisen kriteerit. Kriteereinä voidaan pitää esimerkiksi testauksen aikarajan umpeutumista tai että kaikki testitapaukset on suoritettu 100% kattavuudella ja ilman virheitä [10].



Kuva 11 Testauksen suunnittelu [10]

1. Johdanto
2. Testauksen kohde ja tavoitteet
3. Testausympäristö
4. Testauksen organisointi ja raportointi
5. Testausstrategia ja integrointisuunnitelma
6. Testattavat toiminnot
7. Toimintojen testitapaukset, hyväksymiskriteerit
8. Ei – toiminnallisten ominaisuuksien testaus
9. Erikoistilanteet
10. Ominaisuudet, joita ei testata

Kuva 12 Testaussuunnitelman sisällysluettelo [10]

9.9 Testausmenetelmät

Testausmenetelmä pyrkii kuvaamaan tavan, millä ohjelmistoa testataan. Testausmenetelmän tavoitteena on kattaa testauksen suunnittelu ja ajoitus. Lisäksi sen pitäisi antaa päämäärä ja tarkoitus testauksen suorittamiseen, näitä tarvitaan testausprosessin läpiviemiseksi. Hyvän testausmenetelmän tarkoituksena on tukea ohjelmistoprojektia. Testaustapauksia valittaessa on yleensä käytettävissä vain kaksi päästrategiaa musta- ja lasilaatikkotestaus [43].

Mustalaatikkotestauksessa (engl. black-box testing) testaajalla ei ole tiedossa ohjelmiston oikeaa toimintatapaa entuudestaan, vaan ohjelmiston toiminta päätellään määrittelyiden avulla. Ohjelmaan syötettyjen tietojen ja niistä saatujen vasteiden avulla pyritään lopulta päättelemään, toimiiko ohjelma oikein vai väärin. Lasilaatikkotestauksessa (engl. white-box testing, glass-box testing) tutkitaan koodia, josta pyritään havaitsemaan virheelliset kohdat ja toiminnallisuudet. Edellä mainituista testauksista voidaan myös luoda yhdistelmä, mitä kutsutaan harmaalaatikkotestaukseksi. Harmaalaatikkotestauksessa pyritään käyttämään mustalaatikko- ja lasilaatikkotestauksen parhaat ominaisuudet. Testauksen taso vaikuttaa yleensä testitapausten valintaan, sillä yleensä yksikkötestauksessa käytetään lasilaatikkomenetelmää. Siirryttäessä suurempiin kokonaisuuksiin,

kuten systeemi- ja hyväksymistestaukseen laatikon väri tummuu ja siirrytään kohti mustalaatikkotestausta. Yksiköiden integroituessa kokonaisuudet kasvavat, eikä lasilaatikkomenetelmällä saavuteta haluttua tulosta. Lasilaatikkomenetelmässä yleensä keskitytään yksityiskohtiin, kun taas suuremmissa integroiduissa kokonaisuuksissa halutaan testata toimintaa kokonaisvaltaisemmin. Edellä mainittujen tapojen lisäksi menetelmät voidaan jakaa myös staattisiin ja dynaamisiin (engl. static and dynamic) menetelmiin. Dynaamisessa testauksessa pyritään ajamaan ja käyttämään sovellusta ja näin havaitsemaan virheet. Staattisessa testauksessa ohjelmaa ei suoriteta, vaan testaus perustuu suunnitelman, arkkitehtuurin ja koodin tarkasteluun. [43].

9.10 Testauksen toteutus

Uudistettavaa ohjelmistoa testattiin erilaisilla testausmenetelmillä. Testausta suoritettiin jo kehitysvaiheessa suunnittelemalla ja toteuttamalla yksikkötestejä toteutuksen aikana. Yksikkötestit toteutettiin suunnitellusti jokaisen yksikön toteutuksen yhteydessä. Yksikkö-testejä ei kuitenkaan tehty TDD (test-driven development) -menetelmää hyväksikäyttäen, jossa ensin on tarkoituksena suunnitella ja toteuttaa yksikkötestit ja sen jälkeen vasta suorittaa yksikön toteuttava ohjelma.

TDD-menetelmä ei ole pelkästään testausmenetelmä vaan myös kehitysmenetelmä, jota ohjataan testitapausten avulla. Kehitysympäristöstä pitäisi saada välitön palaute kaikkiin koodimuutoksiin ja kehittäjän pitäisi pystyä suorittamaan testit helposti ja nopeasti kehityksen yhteydessä. Kent Beckin mukaan ohjelmiston pitäisi koostua monista yhtenäisistä toiminnallisuuksista ja sisältää vähän riippuvuuksia, jotta testaaminen olisi sujuvaa ja sitä olisi järkevää suorittaa [44].

Projektin aikana ei käytetty TDD:tä kokonaisuutena, vaan yksikkötestaus toteutettiin perinteisellä yksikkötestausmenetelmällä. Yksikkötestausmenetelmässä ensin suunnitellaan ja tehdään yksikön ja siitä riippuvien yksiköiden koodi, jonka jälkeen valitaan testattavat osiot ja tehdään yksikkötestit. Tähän ratkaisuun päädyttiin, jotta kaikki yksikön osat voidaan testata mahdollisimman vähillä testisijaisten määrällä ja näin saada testit suoritettua ilman kohtuutonta alustus-työtä.

Yksikkötestien lisäksi sovellus oli tarkoitus testata automatisoiduilla käyttöliittymätesteillä, käyttäen Selenium-testityökalua [45] tai muuta vastaavaa työkalua. Tästä kuitenkin luovuttiin, koska SmartGWT-sovellus kehyksen kanssa se ei kaikkien komponenttien osalta ollut mahdollista ilman kohtuutonta vaivaa, vaikka SmartGWT ilmoittaa tukevansa kyseistä testaustapaa. Ongelmia Selenium-testityökalun käyttämisessä aiheutti se, että siinä määritellään testattavat elementit joko elementin nimen tai tunnisteen perusteella, mutta SmartGWT-sovelluskehyksessä ei ole kaikille komponenteille mahdollista antaa tunnistetta tai nimeä. Näin ollen järjestelmätestaus suoritettiin perinteisemmällä menetelmällä ilman automaatiota.

Järjestelmätestaus toteutettiin suunnittelemalla testitapaukset käyttöliittymää mukaillen. Käyttötapaukset suunniteltiin ottamalla käyttöliittymän osio, josta tehtiin testitapaus tai testitapauksia, jotka olivat tarkoitus suorittaa manuaalisesti jokaisella järjestelmätestikierroksella. Toisin sanoen jokainen uusi ohjelmiston kehitysversio testattiin. Kuvassa 13 on esimerkki yhdestä järjestelmän testitapauksesta. Kuvan testitapauksessa testataan sisään-kirjautuminen järjestelmään. Testitapauksesta on mahdollista nähdä testitapauksen prioriteetti, jossa kuvataan, kuinka kriittinen testitapaus on, riippuvuudet muista testitapauksista, testin kulku, testissä mahdollisesti annettavat syötteet, testin suorittamisesta aiheutuva odotettu tulos, mahdollisesti testin jälkeen suoritettavat siivousoperaatiot ja muut tiedot testaajalle testin suorittamista varten.
Suunniteltujen järjestelmätestauksen testitapausten lisäksi sovellusta pyrittiin testaamaan etsivällä testauksella (Exploratory testing), jossa ohjelmaa käytetään satunnaisesti ja pyritään löytämään virheellisiä toimintoja. Sen lisäksi ohjelmaa testattiin vielä virheellisillä syötteillä ja virheellisellä datalla, jolla pyrittiin katsomaan, että ohjelmisto reagoi virhetilanteisiin oikein eikä aiheuta tietoturvariskiä. Osa virheellisten syötteiden testauksista oli jo sisällytettynä järjestelmätestaus vaiheeseen.

S02-TC01: Sisäänkirjautuminen

Luonti aika	20.5.2013
Viimeksi muokattu	-
Prioriteetti ja perustelut	Toiminnon pysäyttävä
Riippuvuudet muista testitapauksista	S01-TC01
Asetukset	Suoritettava testitapaus S01-TC01.
Testin kulku	Kirjoitetaan käyttäjätunnus kenttään hyväksytty käyttäjätunnus ja vastaavasti salasanakenttään hyväksytty salasana
Syötteet	esim. user, user
Odotettu tulos	Ohjelmaan kirjaudutaan sisään onnistuneesti ja luodaan oletus näkymät
Siivous	-
Muut tiedot testaajalle	Oletuksena on, että tietokannan taulusta XXXX löytyy käyttäjä, jolla on tunnukset: käyttäjätunnus: user ja salasana: user

Kuva 13 Testaussuunnitelman sisällysluettelo

10 JOHTOPÄÄTÖKSET

Ohjelmiston vanhetessa sen käyttötarpeet voivat muuttua tai lisääntyä. Muutoksista ja tarpeista johtuen ohjelmistoihin joudutaan tekemään erilaisia muutos-, kehitys- ja muokkaustoimia, jotka ovat omiaan muuttamaan ohjelmiston rakennetta. Rakenteen muuttuminen saattaa johtaa ohjelman monimutkaistumiseen, mikä vaikuttaa ohjelmiston arkkitehtuuriin ja rappeuttaa sitä hiljalleen ajan saatossa. Arkkitehtuurin rappeutuminen taas vaikuttaa ohjelman ylläpidettävyyteen ja kunnossapitoon, mikä saattaa nostaa ohjelman ylläpitokustannukset niin koviksi, ettei sen ylläpito ole enää järkevää.

Tässä diplomityössä on tarkasteltu ohjelmiston uudistamista, testausta ja suunnittelua, kuitenkin pitäen tarkastelunäkökulma kokoajan ohjelman uudistamisessa. Tarkastelun tuloksena voidaan päätellä, että ohjelmiston uudistaminen on aikaa vievää toimintaa ja sille on projektissa varattava riittävästi aikaa ja resursseja, jotta se on mahdollista toteuttaa hyvin. Näin kannattaa toimia ainakin silloin, jos uudistamisen tavoitteena on antaa ohjelmistolle lisää elinkaarta tai mahdollisesti kokonaan uusi elämä. Ohjelmiston uudistamisessa voidaan säästää aikaa hyvällä suunnittelulla, vanhan ohjelmiston ja sen toimialan tuntemisella. Työn tavoitteena oli saada uudistettua tuotantokäytössä oleva luottamushenkilörekisteriohjelmisto. Uudistamisen tarkoituksena oli päivittää ohjelmiston ulkoasua, parantaa hallittavuutta, toimintaa ja saada sovelluksesta verkkopohjainen sovellus ja sitä kautta päästä eroon työasemakohtaisista asennuksista ja riippuvuuksista.

Kaiken kaikkiaan uudistamisprosessissa onnistuttiin odotusten mukaisesti. Projektin aikana valmistui uudistettu ohjelmisto, mikä on hiljalleen syrjäyttänyt vanhaa ohjelmistoa käytöstä. Lisäksi yhtenä onnistumisen kriteerinä voidaan katsoa vanhan käyttäjän käyttökokemus, sillä ohjelma on käyttäjälle osittain tuttu, mutta kuitenkin ulkoasullisesti ja osalta toiminnallisuuksiltaan uusi. Näin käyttäjä säästyy kokonaan uusien asioiden opiskelulta, kun käytössä on tiettyä tuttuutta. Tähän tilanteeseen päästiin, kun ohjelmiston päätoiminnot on pidetty samanlaisena kuin alkuperäisen ohjelman toimintakin oli. Myös tämän kriteeri saatiin projektin aikana täytettyä, jos kriteerin täyttymisen määritelmänä voidaan pitää vanhoilta ohjelman käyttäjiltä saatua positiivista palautetta.

Uudistusprosessin aikana asioita olisi voinut tehdä toisin. Esimerkiksi vanhaan ohjelmistoon ja sen toimintaan olisi ennen uudistusprosessia voinut tutustua vieläkin paremmin, jotta olisi tiedetty kaikki sen toiminnot ja ominaisuudet. Näin toimintojen jatkokehittäminen olisi ollut jo suunnitteluvaiheessa mahdollista. Uudistusprosessin aikana vanhasta ohjelmistosta löytyi osioita, joita ei ohjelmiston rakenteen suunnitteluvaiheessa osannut ottaa huomioon. Tämä vaikutti siihen, että suunnitelmaa joutui muuttamaan melko paljon alkuperäisestä. Suunnitelman muuttaminen saattoi aiheuttaa muutoksia

tulevaan tai jo tehtyyn ohjelmiston rakenteeseen. Näistä ongelmista ja asioista kuitenkin selvittiin hyvin.

Projektin jälkeen jäi vielä parannettavaa, esimerkiksi joidenkin toimintojen vasteaikoja pitäisi saada parannettua. Parannusta kaivataan varsinkin silloin, kun ohjelman käsittelemä tietomäärä kasvaa erittäin suureksi. Lisäksi ohjelmiston käyttö on tuonut jatkokehitystoiveita ja kehitystoiveita joidenkin toimintojen osalta. Ohjelmaa tullaan vielä jatkossa kehittämään niin omasta toimesta kuin asiakkailta saadun palautteen mukaisesti. Lisäksi tulevaisuuden tavoitteena on rakentaa kolmansia osapuolia hyödyttävä rajapinta sovelluksen tietoihin.

Kaiken kaikkiaan uudistamisprosessin aikana opittiin paljon toimialasta ja toimintaympäristöistä, johon ohjelmisto on alkujaan suunniteltu. Lisäksi opittiin ohjelmiston uudistamisessa vaadittavista toimenpiteistä ja työmääristä, sekä työmäärästä aiheutuvan suunnittelun ja taustatyön tärkeydestä. Projektin aikana oli mahdollisuus myös tutustua yleisellä tasolla ohjelmiston uudistamiseen. Näiden asioiden ohella opittiin myös uusi ohjelmistokehys, jota ohjelman toteutuksessa käytettiin. Kaiken kaikkiaan projekti vaati paljon, mutta myös antoi ja opetti paljon.

LÄHTEET

[1] CGA. [WWW]. [Viitattu 29.10.2015]. Saatavissa: http://fin.afterdawn.com/sanasto/selitys.cfm/cga

[2] Merkkaus- ja sivunkuvauskielet. [WWW]. [Viitattu 29.10.2015]. Saatavissa: http://users.metropolia.fi/~pekkasah/merkkaus/etusivu.html

[3] DBPool : Java Database Connection Pooling. [WWW]. [Viitattu 29.10.2015] Saatavissa: http://www.snaq.net/java/DBPool/

[4] EGA. [WWW]. [Viitattu 29.10.2015]. Saatavissa: https://fi.wikipedia.org/wiki/EGA

[5] Productivity for developers, performance for users. [WWW]. [Viitattu 29.10.2015] Saatavissa: http://www.gwtproject.org/

[6] Frank Boumphrey, Cassandra Greer, Dave Raggett, Jenny Ragget, Sebastian Schnitzenbaumer, Ted Wugofski, 2001, XHTML. Helsinki: Oy Edita Ab

[7] Matti Vuori, ISTQB:n testaussanasto. 2007. [WWW]. [Viitattu 4.3.2015]. Saatavissa http://fistb.ttlry.mearra.com/sites/fistb.ttlry.mearra.com/files/istqb_sanasto.pdf

[8] Tero Ahonen, Tapio Hämeen-Anttila, Kim Åstrand, 2003, JavaServlets. Jyväskylä: Docendo Finland Oy

[9] Suunnitteluprosessista. [WWW]. [Viitattu 14.4.2013]. Saatavilla: http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/030308/1146204519802/1146224777754/1146226151084/1146226538347.html

[10] Haikala, I. & Merijärvi, J. 2004. Ohjelmisto Tuotanto, Hämeenlinna: Talentum Media Oy.

[11] Tietosuojaseloste. [WWW]. [Viitattu 10.2.2013]. Saatavissa: http://www.tampere.fi/material/attachments/t/6BgAdBFhM/tietosuojaselosteluottamushenkilorekisteri.pdf

[12] Rekisteriseloste. [WWW]. [Viitattu 29.10.2015]. Saatavissa: http://www.tietosuoja.fi/fi/index/useinkysyttya/rekisteriseloste.html

[13] SmartGWT quick start guide. [WWW]. [Viitattu 11.5.2015]. Saatavilla: http://www.smartclient.com/releases/SmartGWT_Quick_Start_Guide.pdf

[14] Ilkka Haikala ja Tommi Mikkonen, P 2011. Ohjelmistotuotannon käytännöt, Helsinki 201, Talentum.

[15] About the JFC and Swing. [WWW]. [Viitattu 29.10.2015]. Saatavissa: http://docs.oracle.com/javase/tutorial/uiswing/start/about.html

[16] Test-driven development as a defect-reduction practice 17-20.11 2003. [WWW]. [Viitattu 7.3.2015]. Saatavissa: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1251029&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1251029

[17] VGA. [WWW]. [Viitattu 29.10.2015]. Saatavissa: http://fin.afterdawn.com/sanasto/selitys.cfm/vga

[18] Visual Basic. [WWW]. [Viitattu 29.10.2015]. Saatavissa: https://fi.wikipedia.org/wiki/Visual_Basic

[19] Ari Rantala, 2005, Web-Ohjelmointi. Jyväskylä: Docendo Finland Oy

[20] Allworth S.T. & Zobel R.N. 1991. Introduction to Real-time Software Design. 2nd edition. London: MacMillan education ltd.

[21] Lehman's laws of software evolution. [WWW]. [Viitattu 7.4.2015] Saatavissa: http://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution

[22] Henkilötietolaki. [WWW]. [Viitattu 9.2.2013]. Saatavissa: http://www.finlex.fi/fi/laki/ajantasa/1999/19990523.

[23] Henkilörekisteristä lain kannalta [WWW] [Viitattu 9.2.2013] Saatavissa: http://www.cs.tut.fi/~jkorpela/hlorek.html

[24] Luottamushenkilön asema. [WWW]. [Viitattu 10.2.2013]. Saatavissa: http://www.kunnat.net/fi/asiantuntijapalvelut/laki/hallintojuridiikka/kunnan_hallinto/luottamushenkilon-asema/Sivut/default.aspx

[25] Luottamushenkilörekisteri. [WWW]. [Viitattu 10.2.2013]. Saatavissa: http://www.raisio.fi/osallistu-ja-vaikuta/luottamushenkiloiden-infopankk/fi_FI/luottamushenkilorekisteri/

[26] Käyttöliittymäsanoja. [WWW]. [Viitattu 9.3.2013]. Saatavissa http://www.ttlry.fi/k%C3%A4ytt%C3%B6liittym%C3%A4sanoja

[27] User interface. [WWW]. [Viitattu 9.3.2013]. Saatavissa http://en.wikipedia.org/wiki/User_interface

[28] Komentokäyttöliittymä. [WWW]. [Viitattu 9.3.2013]. Saatavilla: http://fi.wikipedia.org/wiki/Komentoliittym%C3%A4

[29] Tekstipohjainen käyttöliittymä. [WWW]. [Viitattu 9.3.2013]. Saatavilla: http://fi.wikipedia.org/wiki/Tekstipohjainen_k%C3%A4ytt%C3%B6liittym%C3%A4

[30] Käyttöliittymien varhainen kehitys. [WWW]. [Viitattu 16.3.2013]. Saatavilla: http://www.cs.helsinki.fi/u/kerola/tkhist/k2000/alustukset/kayttoliittymat/seminaari.html#graafiset

[31] Graphical User Interface. [WWW]. [Viitattu 16.3.2013]. Saatavilla: http://cs.joensuu.fi/~jimmonen/gkl_moniste/gkl_v202.html

[32] Käyttöliittymäsuunnittelun periaatteita. [WWW]. [Viitattu 14.4.2013] Saatavilla: http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/030308/1146204519802/1146224777754/1146226104890/1146226324883.html

[33] Mary Beth Rosson and John M. Carroll, P. 2002. Usability Engineering, Imprint: Morgan Kaufmann.

[34] Wille Kuutti, 2003. Käytettävyys, suunnittelu ja arviointi, Saarijärvi: Talentum Media Oy ja Wille Kuutti.

[35] Nielsen, Jacob 1993. Usability engineering: Sanfrancisco (CA) Academic Press

[36] (Norman, 1998) Powell, Thomas A., P. 1998 Web Site Engineering: Beyond Web Page Desing. Prentice Hall

[37] Käytettävyys käyttöliittymäsuunnittelussa. [WWW]. [Viitattu 27.4.2013]. Saatavilla: http://www.cs.tut.fi/~grako/2008/luennot/grako_kayt_luento.pdf

[38] Harsu Maarit, P 2003. Ohjelmien ylläpito ja uudistaminen. Jyväskylä, Gummerus Kirjapaino Oy.

[39] Server Communication. [WWW]. [Viitattu 11.5.2013]. Saatavilla: https://developers.google.com/web-toolkit/doc/latest/DevGuideServerCommunication?hl=fi-FI

[40] Kai Koskimies, Tommi Mikkonen, Ohjelmistoarkkitehtuurit. Talentum 2005. [WWW]. [Viitattu 20.2.2015]. Saatavissa: http://www.cs.tut.fi/~ohar/OHAR2012_13-KIRJA-KoskimiesMikkonen.pdf

[41] Smart GWT showcase. [WWW]. [Viitattu 22.3.2015]. Saatavissa: http://www.smartclient.com/smartgwt/showcase/

[42] Mika Katara, 2011. Ohjelmistojen testaus, Tampereen teknillinen yliopisto: Ohjelmistotekniikan laitos.

[43] Tiina Tersa, TESTAUSMENETELMIEN KÄYTTÖ SOVELLUKSEN SYSTEEMITESTAUSVAIHEESSA. 2002. [WWW]. [Viitattu 4.10.2015]. Saatavissa http://www.mit.jyu.fi/opetus/opinnayte/gradu/systeemitestaus/Tiina_Tersa.pdf

[44] Kent Beck. Test-Driven Development: By Example. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[45] SeleniumHQ Browser Automation. [WWW]. [Viitattu 9.11.2015]. Saatavissa: http://www.seleniumhq.org/